Salesforce *Governance*

CloudKettle

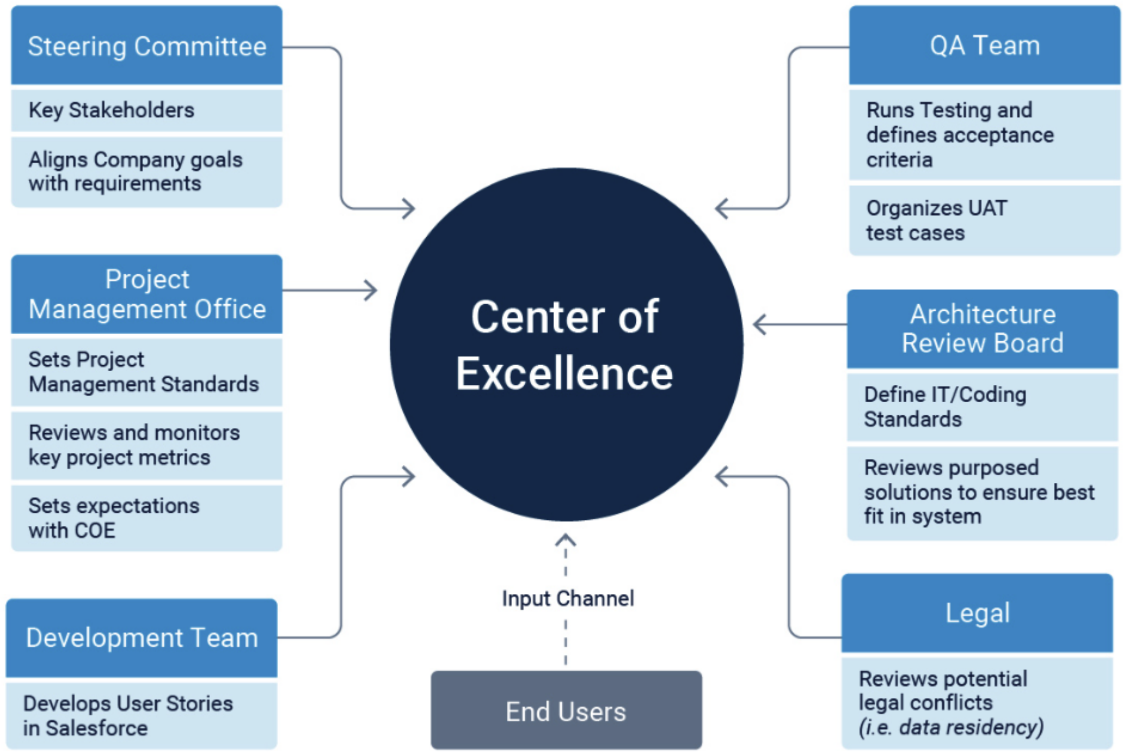
Table of Contents

Introduction	1
Salesforce Center of Excellence	2
Release Management Policy	10
Technical Debt Policies	18
Data Governance	19
Metadata Governance	21
Integration Policy	27
Code Standards Documentation	29

The following document provides a general high-level overview of best practices for Salesforce Governance. This is not a comprehensive guide or recommendation of a solution tailored to your organization's needs. It is intended as a guideline and as a quick starting point to developing a Governance Policy. The sections following the Salesforce Center of Excellence will provide further insight into certain subsections of the Salesforce Center of Excellence.

Salesforce Center of Excellence

CloudKettle recommends that all organizations leverage a Salesforce Center of Excellence (COE) team structure for the best success with Salesforce projects and ongoing support of the Salesforce org. Having a solid structured approach to monitoring Salesforce releases will be critical when considering the future of company acquisitions by an organization, and how existing employees at acquired companies fit into the central team responsible for all Salesforce orgs. The following illustration outlines the recommended structure to use:


Steering Committee
Key Stakeholders
Aligns Company goals with requirements
Project Management Office
Sets Project Management Standards
Reviews and monitors key project metrics
Sets expectations with COE
Development Team
Develops User Stories in Salesforce
Center of Excellence
Input Channel
End Users
QA Team
Runs Testing and defines acceptance criteria
Organizes UAT test cases
Architecture Review Board
Define IT/Coding Standards
Reviews purposed solutions to ensure best fit in system
Legal
Reviews potential legal conflicts
(i.e. data residency)

Steering Committee

The steering committee is made up of executive users who define the high-level requirements for changes made to the system. The steering committee's responsibilities are to ensure that Salesforce functionality supports the company's overall goals; they lead the COE from a high-level perspective.

Project Management Office

The project management office (PMO) governs and communicates the overall project structure, and controls and owns all Project Management (PM) tools such as Jira, Burndown Charts, etc. PMs as part of the PMO are responsible for communicating project deadlines, working closely with the development team, identifying and trying to alleviate blockers and ensuring an up-to-date project schedule based on the burn down rate of current requirements.

In an agile release schedule PMs are responsible for creating sprints based on approved business requirements. When deciding how much should go into a sprint, PMs review the estimated length of stories and assign resources that are available for that sprint to the tasks which need to be completed.

Quality Assurance Team

The quality assurance (QA) team works to ensure that thorough testing is done during the development process and orchestrates a full end-user testing phase with the help of the PM. QA sets standards for releases being delivered, defines what metrics need to be met before accepting a story as completed, and executes on the actual test scripts to give a non-biased test on work being done by the development team. QA can help support automated testing, but within the context of Salesforce, automated tests to support Apex code should be the responsibility of the development team.

Architecture Review Board

The architecture review board reviews proposed solutions by the development team and ensures they align with the overall system design to ensure a scalable Salesforce ecosystem. They ensure all best practices are being followed when new projects are being proposed. The review board could also be responsible for defining IT standards such as how business and technical solutions should be formatted, as well as coding standards that must be followed to ensure consistency in the ecosystem. Sometimes these design best practices are broken out into a separate team referred to as a “design authority”. However, in some cases, depending on tea size, the architecture team may take responsibility for the design best practices.

Development Team

The development team is responsible for building the end product and creating the solutions to particular business problems. The development team will often lean heavily on the architecture review board for help on the best practices for design to help pass the board's approval on complex solutions. The development team is often the largest team as this is the team executing on the business requirements. They help give estimations to the project management team to forecast how long a total sum of work will take. The development team can also support the overall dev ops process for how automated deployment and testing (CD/CI) is configured. However, this responsibility could be abstracted to a specific "DevOps team" if the complexity of managing multiple pipelines becomes too high.

Legal

The legal team ensures that all regulatory and compliance needs are met and is responsible for bringing up any potential problems with regulation that are known and mitigated before a project moves forward. Having legal participation in the COE is extremely important with the growing number of data governance laws that are expanding in the global ecosystem.

End Users

The end users are the ones actually using the system. Although they would not be a direct member of the COE and should not be involved in the day-to-day operation, their input is critical. Ultimately, it is their satisfaction with Salesforce that will define the adoption rate of the product. Helping increase the productivity and functionality of the end users should always be a priority and any goals driven by the steering committee should have that in mind. To get end user feedback it is important to have a way for them to either fill out recurring surveys, submit cases or issues they see, or submit ideas they may have. Interviews with end users once a year will help surface problems that can later be turned into projects to deliver a better product.

Internal Releases

To manage the process around deploying custom changes developed in-house, organizations should adopt a strict Internal Release Protocol that tracks the entire process from user requests and requirement gathering, to the creation and completion of test cases.

Implementing new metadata components such as new Process Builder automation or Validation Rules can have a serious impact if developed and tested directly in Production. To ensure any changes made do not interrupt business operations, all internal development should be completed in a sandbox environment and tested thoroughly before being deployed to a production environment.

Internal Releases should be on a set bi-weekly or monthly schedule, ensuring that the Administrator has sufficient time to analyze the change requests, determine the best approach, and test the feasibility in a sandbox environment. Enforcing a release schedule safeguards the Administrator from creating quick-fixes on the fly, and sets the expectation around Salesforce Change Management for the end users.

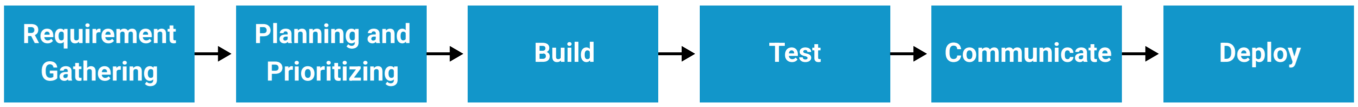
Salesforce Releases

As a component of the Center of Excellence, a dedicated Salesforce Admin at the organization should be tasked to read and review every new Salesforce release to be aware of new features and functionality that will be available. Staying on top of the releases will help mitigate any effects critical updates may have on a Salesforce org.

Salesforce Deployment Process

A Salesforce release management strategy increases productivity and deployment velocity while decreasing costs and downtimes due to changes in the system.

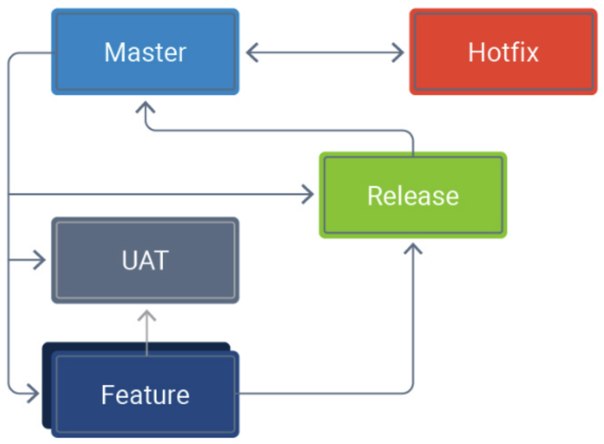
A Salesforce Release Management Policy is necessary to support a well-defined release management process that allows organization to streamline changes from development to production. CloudKettle recommends following the Management overview as shown below:



Salesforce Sandboxes

Salesforce allows for the refreshing of sandboxes, which wipes all existing data in a partial sandbox and creates a new version that replicates what is in the production org. As a best practice, CloudKettle recommends using a partial copy sandbox as part of the company’s deployment strategy. Having the partial copy sandbox that is untouched and always kept in sync with production provides an environment with a good amount of data and the same metadata configuration as production. This is an ideal environment to test hotfixes before they get pushed to production.

The following chart gives a visual representation on how to map a GIT strategy to various Salesforce environments. Ideally this would use the automated deployment process described below:



BRANCH TO ORG MAP

Feature	Developer Sandbox
UAT	Full Sandbox
Hotfix	Partial Sandbox
Master	Production

Version Control

Salesforce does not have built-in version control on its platform. Once any form of work is deployed it immediately overrides the previous metadata that existed on the platform without much of a historical trace of it ever being there. If the org was ever customized in a way that stopped or halted the ability to make sales or provide support, the only way natively revert to a previous version would be to manually track down any changes made that may have caused this stoppage and work backwards to try to resolve the issue.

CloudKettle recommends leveraging either Github or BitBucket as an external version control system. Having a version control system in place allows for the use of continuous deployment and continuous integration which simplifies the deployment process for every feature and also ensures for a significant reduction in errors as full retention tests can be easily run for every change made to the org.

Continuous Deployment

Continuous Deployment (CD) represents the process of deploying changes directly from a version control system to your Salesforce environments. One of the most overlooked components of Salesforce development is the time it takes to deploy changes. Deployment can cause errors based on dependency conflicts, testing errors, forgetting to add components, or Salesforce issues that are unpredictable. CD in combination with Continuous Integration (discussed below) help identify these potential problems far ahead of time and provide a system that reduces them from happening in the first place.

The tooling that is used to support CD depends on preference and potentially the version control system used, as BitBucket and Github both have their own CD/CI tools built into their product. There are third party tools such as Jenkins that are popular for this purpose but CloudKettle would highly recommend using either BitBucket's Pipelines or Github Actions to support the automation based on the triggering event of merged pull requests.

Continuous Integration

Continuous Integration (CI) represents the process of running automated tests and automated deployment verifications at either a given frequency or in real time when changes are made to a version control system. CloudKettle recommends the best practice of running CI against the org that is going to be merging every time a pull request is created or updated (new commits). For example, if building in a feature branch the next org to push to is the UAT org. When a pull request is created to move the feature to the UAT branch, there should be validation in place that all Apex tests and Jest tests (LWC Testing) need to pass. There should also be deployment validation that succeeds in order to be allowed to merge the pull request from feature to UAT. This ensures that new features can be assessed in development and potential negative impacts can be reviewed before updates are deployed. This significantly reduces development time, as it will reduce bugs and make them known while development is still ongoing.

Release Cadence

The release cadence should have a set of meetings that ensure all required stakeholders have the appropriate understanding of the scope of each deployment. A deployment will happen every other week or once a month based on how often an organization plans to deploy any enhancement or an upgrade.

The deployment meeting will be held to mitigate business impact of unforeseen errors or issues during a release. It is the responsibility of the Salesforce Admin/Release team to communicate the deployment window to all users and ensure all required stakeholders have the appropriate understanding of the scope of each deployment. CloudKettle recommends deployments occur on Tuesday, Wednesday, or Thursday to allow for resources to be available to help troubleshoot if issues arise. For guidance on structuring the Release Management Process, see CloudKettle's Release Management Policy Template.

Salesforce Documentation Standards

Org Handbook

CloudKettle recommends that in addition to the documentation of releases, an Org Handbook should be maintained.

An Org Handbook documents the entirety of the Salesforce org and is used to maintain institutional knowledge of systems and processes. At a minimum the handbook is recommended to cover the following items:

- Org Architecture Diagram
- Data Flow Diagram
  - Integrations, webforms and all other sources of data being transferred in an out of Salesforce
- MQL and SQL processes for the org
- Lead conversion & Sales Processes
- High level summary of Automation and processes in the org
  - For complex and multi-stage processes, explain the interaction and expected outcome/use case of the automation
- High level summary of the use case for all third party integrations

Incident Reporting Logs/Policy
In the event of unplanned downtime or feature failure, etc an incident response and reporting policy needs to be drafted.

The policy will cover steps to triage the issue and a contact chain to action the issue based on severity and the timing of the incident to assure that the appropriate person is contacted to solve this issue.

Once the incident is resolved, a retrospective should occur in which the incident is reviewed and the sequence of events is documented and reviewed to identify gaps in process or issues that lead to the incident in question. This meeting should include stakeholders, project managers and Salesforce Team members.

Incident reporting is never to be treated as a disciplinary, or tribunal to assign blame. Incidents are not uncommon and often under-reported or swept under the rug. It is important to foster a culture where incidents are treated as a learning experience and studied to improve existing processes. Incidents that are not studied and reviewed will likely occur again.

Release Management Policy

General Recommendations

The appropriate Release Management Policy will be dependent on the level of complexity and the needs of your org. For example an org where custom Apex Classes and triggers will require some sort of code repository for version control, whereas an Org that uses the Flows and Clicks not Code functionality can maintain a Release Model using in board Salesforce tools.

At a minimum CloudKettle recommends flowing a Sandbox Management process such as the one covered in the previous Sandbox Section. Upon deciding on a Sandbox and deployment process, adopting Salesforce's Dev Ops Center to track sprints will formalize and apply structure to the policy. In addition, Deb Ops Center will maintain versioning of Salesforce MetaData for approval and promotion to the next Sandbox.

Business Requirements Documentation and Product Backlog

Users should leverage a business requirement document to request additional functionality in Salesforce. The document needs to be templated and provide a guideline to creating a ticket/feature request.

There should be two templates, one for a bug report and another for a new feature request.   The document should contain the following:

Bug Report:

- Details of the request
    - When did the issue occur?
    - Steps to replicate the issue
    - Screenshot of Error/issue
- The business impact of the issue?
    - Is there a work around?

Feature Request:

- Functional details of the request
  - New feature or Improvement?
  - What is the desired outcome of the changes?
- The business impact of their request (including justification of importance)
  - What pain-point does this solve?
  - Does this request save time/simplify an existing process?

These documents should be readily available to assist users, and an email account, such as salesforce@[yourcompanyname].com should be set up to receive all requests. Alternatively, cases or an external ticketing system can be used to capture the requests. CloudKettle strongly recommends that an issue & project management tool such as Jira is used to track, prioritize and action these requests.

These requests will make up the 'Salesforce Product Backlog' and will be 'groomed' before entering any sprint and being worked on by the Salesforce team.

The Salesforce product owner's responsibility is to prioritize the items in the 'Salesforce Product Backlog' by working with the various stakeholders and ensuring that the requirements are well documented. Each item in the backlog must have an effort and priority assigned to it to size the tasks.

Sprint Planning and Scope

Before each sprint, there will be a sprint planning session where priority items in the 'Salesforce Product Backlog' are properly groomed and added to the sprint backlog.

Once the scope of the sprint is decided, items should not be added unless necessary. The product owner or scrum master's responsibility is to ensure there is no unnecessary scope creep. In only critical circumstances should items be added to an ongoing sprint.

This practice can be time-consuming at first but is critical to understanding departmental needs and planning for future success. Operating within a 'Salesforce Product Backlog' and sprints will also allow all incoming requests to be prioritized and visible across the organization. For more information on this topic, see How to Create a Salesforce Action Plan.

Tips for Success

Organizations often start with an informal process for requesting Salesforce changes (email requests, instant messaging a Salesforce administrator, etc.). However, this is not sustainable long-term and will often lead to overlapping functionality and a slower velocity of changes made.

For optimal results, a standardized process must be documented, followed, and be championed by the leadership team. The process should include these essential steps:

1. Make formal requirements gathering document
2. Prioritize the requirements in a project management tool
3. Judge the effort needed for all requirements
4. Have a planning session to determine what will be worked on during the next sprint
5. Work on a set of requirements and only those requirements for each sprint
6. Inform all stakeholders of what is being worked on in each sprint
7. Business Requirements Document Template

The following table can be used as a starting point for the Business Requirements document.

Question	Response
Requestor Name:	
Requestor Department:	
Date of Request:	
Type of Request (Bug, Enhancement, Question, Maintenance):	
Description of Request: (note pain point with the current process, if applicable)	
Current Functionality:	
Business Value:	
Business Impact (High, Medium, Low):	
Business Impact Justification:	
Current workarounds (If applicable):	

Salesforce Sandboxes

Salesforce subscriptions come with the ability to create “Full” and “Partial” Sandboxes that are recreations of the full production instance of Salesforce, but with either a complete (“Full”) or partial (“Partial”) replica of data. Depending on the Organization’s Edition and agreement with Salesforce, the quantity and availability of Sandbox will change or be unavailable. These two types of sandboxes will contain data about customers and prospects.

Should privacy and access to this data be a concern, A license for Salesforce Data Mask can be purchased to anonymize sandbox data. For more information on this topic, see Salesforce’s documentation.

Salesforce allows for the refreshing of sandboxes, which wipes all existing data in a Full or Partial sandbox and creates a new version that replicates what is in the production instance. At that time, any data that was deleted from the production instance will cease to exist in the sandbox.

A example of a company's sandbox refresh policy is as follows:

Type	Name	Use	Refresh
Production	CK Sample Org	• Live Environment	
Full Sandbox	CKUAT	• User Acceptance Testing • Staging • Performance Testing	Insert full sandbox refresh period (begin with 6 months and reduce)
Partial Copy Sandbox	CKHotfix CKPartial CKTraining	• Training • 'hotfix' testing	30 days
Developer Pro Sandbox	IntegTest	• Merge changes from all developer sandboxes • Integration Testing	After every major release
Developer Sandbox	CPQTesting SimoneDev	• Development and testing in an isolated environment	After every major release

Hotfixes

A best practice we share with our clients is to use a partial copy sandbox as part of the company's deployment strategy. Having a partial copy sandbox that is untouched and always kept in sync with production provides an environment with a good amount of data and the same metadata configuration as Production. This is an ideal environment to test 'hotfixes' before they get pushed to production.

A “hotfix” is a bug or issue found by users from the latest release that can’t wait for the next release to be fixed and therefore requires immediate attention. These are often found in the middle of the next release cycle, meaning the sandboxes are out of sync with production. However, developers can work on the fixes in the partial copy sandbox, as it is a metadata replica of production, and not have to worry about any conflicts from the current sprint.

Release & Meeting Cadence

An essential requirement of a strong release management strategy is establishing a release cadence. The release cadence should have a set of meetings that ensure all required stakeholders have the appropriate understanding of the scope of each deployment. It is best to do deployments in the middle of the week and not on Fridays as issues may come up during the weekend and there will be limited resources to help troubleshoot. Typical cadences are bi-weekly or monthly depending on the organization's needs.

Below is an example of a bi-weekly cadence:

A deployment will happen every other week, at an agreed upon time by all stakeholders. A bi-weekly deployment meeting will be held to mitigate the business impact of unforeseen errors or issues during a release. It is the responsibility of the Salesforce Product Owner to communicate the deployment window to all users and ensure all required stakeholders have the appropriate understanding of the scope of each deployment. Deployments will occur on Tuesdays, Wednesdays, or Thursdays to allow for resources to be available to help troubleshoot if issues arise.

To support the release cadence, a few meetings and artifacts are needed:

- A log of all the proposed changes will be maintained
- Two business days before deployment, there will be a meeting to discuss the ready changes, any dependencies, and manual steps that need to be undertaken
- After this meeting, the log would be submitted to the Salesforce Deployment Lead. No new items can be added to the release’s deployment after this point. All those who have access to make changes to the system must be notified of all the components in the deployment.

	M	TU	W	TH	F	M	TU	W	TH	F	M	TU
Build	🟦	🟦	🟦	🟦	🟦							
Deployment to Merge (as build completes)	▨	▨	▨	▨	▨							
Unit Testing/QA in Merge	⬛	⬛	⬛	⬛	⬛							
Deployment to Full Sandbox						🟦						
Performance and Regression Testing							🟥	🟥	🟥			
User Acceptance Testing							🟫	🟫	🟫			
Deployment Meeting										🟪		
Sign-off on log											🟧	
Deployment to production		🟩										🟩

The figure above is a schedule of a two-week sprint in which:

- The build is scheduled to be completed by the end of the first week,
- The second week is spent on deployments to the full copy sandbox, testing, and bug fixes related to the development.

If there are issues that arise during performance testing and UAT, the developers will correct these issues in their development sandboxes, promote the changes as per the standard process and re-submit the functionality for UAT until it passes.

User Acceptance Testing (UAT)
End-users will test the functionality in a sandbox environment before it is deployed to production. Users will be given a document that outlines the steps they need to execute and will provide any feedback regarding the test cases through this document.

Communication Plan
As part of the deployment process, the Salesforce Product Owner will announce a maintenance window well in advance in which users will be unable to access the system.

Additionally, if required, Salesforce Product Owner will schedule training with the appropriate users to prepare them for new functionality.

Release Notes

Release notes are an effective way to communicate changes that have been made to the system.

To conclude the Sprint, Salesforce Product Owner will assure there is a release notes document containing any bug fixes or functionality updates created.

To be effective, release notes need to be written in an easy to understand way. Release notes for internal users can consist of text, screenshots of the system and links to video recordings demonstrating new functionality. They should not be technical, but rather highlight the effects of the changes for end users, highlight new features or changes in process.

Technical Debt Policies

Managing technical debt in Salesforce is a challenge almost all enterprises face. Steep technical debt in Sales and Service Cloud affects Salesforce performance, the accuracy of data, and the reliability of Salesforce processes.

Technical debt extends beyond code and includes abandoned or overlapping processes, workflows, custom fields, etc. Technical Debt is classified as either incurred or evolved. Incurred technical debt is deliberately or inadvertently accumulated; like using an obsolete method of solving a problem. For example, using a Salesforce feature that is on the roadmap to be retired.

Evolved technical debt is created over time as the result of changes both in Salesforce's platform and your organization's Salesforce org. For example, when an organization has to retire a solution because the number of Salesforce contacts has grown significantly; making the historic solution obsolete.

Having a technical debt reduction policy is a necessary component for decreasing technical debt and improving data integrity. This template was built for enterprise teams and should be populated with your organization's relevant information in the highlighted areas.

To review our recommendations on Technical debt, please see our blog post on technical debt and review the attached Technical debt reduction policy.

Data Retention and Destruction

As the hub of customer data, Salesforce is a key consideration in security and privacy compliance frameworks. Having a Data Retention and Destruction Policy is necessary to comply with many regulatory frameworks, including; SOC 2 and ISO.

It is also best practice for enterprises to minimize the amount of data kept in Salesforce in case the organization ever experiences a breach. By regularly deleting data from Salesforce, it will help mitigate both the liability and impact of any potential breaches. Lastly, having a Data Retention and Destruction Policy helps to improve Org speed and user experience by eliminating unneeded data.

CloudKettle provides a template for data retention and destruction that can be found here: CloudKettle's Salesforce Data Retention and Destruction Policy

Data Quality

A data quality audit should be conducted on a regular basis to ensure that the data in the org is actionable, clean and accurate. Depending on the integrations and customization of your Org, the additional items may need to be audited.

This audit would include using tools such as Field Trip to review the population of fields, and review minimally populated fields (ie: those populated on < 5% of records) for removal/deprecation or additional training for adoption.

This should include reviewing existing fields for chances to consolidate text fields into Picklists to provide reportable buckets, opportunities to use mandatory fields, field dependencies, or validation rules to ensure records always contain at least the minimal amount of data to be actionable.

Once the Fields are reviewed, Admins should proceed to look at the records - particularly on high volume objects such as Accounts, Contacts and Leads - with the goal to identify and merge duplicate records. The effectiveness of current Duplicate and Matching Rules or third party routing/deduplication tools should also be reviewed to ensure alignment with current processes and business logic.

Admins should review all data ingestion points for the org. This data should be standardized across integrations and areas for review should include: deduplication, ensuring consistent picklist values, and ensuring only required and actionable data is coming into the org. For ingestion points such as list loads, ensure that standardized templates are created and maintained.

Metadata Governance

Metadata Governance covers standards for your organizations for documentation and naming conventions with Salesforce. Having description fields and proper naming conventions simplifies maintenance and onboarding time of new team members to the organization. Below are best practices and general guidelines for metadata governance:

General Guidelines

1. If there is a description field, it should be completed.
2. The Description must include the purpose of the item in question
3. If the component is a helper field for any automation or a requirement for an integration, a line should be included to indicate where it is used and why
   a. Integration Example: Used to hold the Contact Signer Look-up for DocuSign Envelope Template "Approved Quote for client Signature"
   b. Flow Example: Helper field to calculate the ARR for the Flow: Total ARR on Account.
   c. A component installed by a package may be managed and thus the description and names may not be editable. If the functionality is extended, document this externally.
      i. If the component is not managed, avoid changing the name/API name to prevent confusion, update the descriptions as required but differentiate between the original unchanged description, and custom newly added descriptions
4. API Names must always match the Labels
5. Spelling and Grammar mistakes in Names are unacceptable. Speed should not take precedence over this. If a mistake is noticed, update it before moving onto another task.
6. Descriptions & Names should follow proper writing guidelines. Capitalization, full sentences and no abbreviations.

In general, the names of fields and functionalities must be descriptive, and approved by the Client. If one is familiar with the org, one should know the purpose by reading the name.

Fields

Formula Fields

1. Writing Formulas: Logic in formulas AND/OR Statements should be indented according to K&R Style or another agreed upon style.
2. Comments should be used to explain functionality of any REGEX or complex and not immediately parseable lines in the formula. This comment is to provide the context necessary to understand the formula. Specifically in formulas with multiple outcomes and constraints.
    a. Ie: /* <Comment> */
    b. If the formula logic is complex, but inline comments are not beneficial to the explanation, using the description field is an acceptable alternative
3. If an update to an existing formula is required, add a date in the description and include the changes made and an "Updated By <NAME>" Tag.

Regular Fields

1. Naming Conventions: Names of fields must be descriptive, and approved by the Stakeholders. If one is familiar with the Org, one should know the purpose by reading the name. If this is not possible, recommend a tool-tip for further clarification.
2. The Description must include the purpose of the field
3. The Description must include "Created by <Name>"
4. If the Field is a helper field for any automation or a requirement for an integration, a line such as the following should be included
    a. Used to hold the Contact Signer Look-up for DocuSign
    b. Helper field to calculate the Automotive Revenue for the Flow: Total Revenue.

Custom Objects

Custom objects should be created to extend the functionality of the core objects. Where possible, CloudKettle strongly recommends that the Standard Objects be used first and the architecture and data model of Salesforce be preserved.

Creating new custom objects should be reviewed by the architecture review board to make sure they conform to the Salesforce Data model and the current architecture of the org.

1. Naming Conventions: Names of Objects must be descriptive, and approved by the Stakeholders. If one is familiar with the Org, one should know the purpose by reading the name.
2. The Description must include the purpose of the Object.

Automation

Flows

Flow Level Name and Descriptions:

1. Naming Flows should follow this convention: <Object Name>: <Descriptive Title>
    a. Opportunity: Submit for Approval When Stage and Data Requirements Met,
    b. The description should then read:
        i. Submits the Opportunity for Approval when Stage = Negotiating and Amount > 0 & Discount Requested = True. Created as part of the Deal Desk Refresh project. Created by Name.
    c. If the Automation is updated in the future update the description
        i. 25/01/2023 – Updated by <Name> to include New criteria Owner <> IntegrationUser Submits the Opportunity for Approval when Stage = Negotiating and Amount > 0 & Discount Requested = True. Created as part of the Deal Desk Refresh project. Created by <name>

Node Level Descriptions:

1. All nodes must have a description that describes the functionality of the node. An admin familiar with flows must be able to read the description and understand the purpose of the node and how it fits into the automation as a whole
2. If you are building a new Flow or conducting a major overhaul of an existing flow, Created by “Name” is not required at this level, HOWEVER if you are providing a fix or a minor to an existing flow, it is recommended to add “Updated by <Name>”.

Flow Resource Names

1. Names of variables/formulas/ must be descriptive. If one is familiar with the Org, one should know the purpose by reading the name.
2. Descriptions must be populated and explain the purpose of the Resource. An admin familiar with flows must be able to read the description and understand the purpose of the resource and how it fits into the automation as a whole.
3. If you are building a new Flow or conducting a major overhaul of an existing flow, “Created by <Name>” is not required at this level, HOWEVER if you are providing a fix or a minor to an existing flow, it is recommended.

Custom Flow Errors Messages

1. Admins should use the Custom Error Message element (Flow API Version 59.0+) for before-save and after-save record triggered flows, to display error messages to explain what went wrong or how to correct it.

Process Builders/Workflows

1. No new Process Builders or Workflow Rules should be created in client orgs, as this functionality is being phased out in favor of Flows.
2. If you need to update an existing Automation, add a date in the description and include the changes made and an “Updated By <Name>” Tag.

Change Sets

1. Change Sets are to be created in tandem with a Configuration Workbook (CWB) to document the components (Using DevOps Center can simplify this process).
2. Change Set names should include a reference to the current project as a descriptive name
3. Change Set descriptions should include a summary of what is being pushed, as well as a “Created By <Name>” Tag.

Validation Rules

1. Writing Validation Rules: Logic in formulas or AND/OR Statements should be indented according to K&R Style.
2. Comments should be used to explain functionality of any REGEX or complex and not immediately parseable lines in the Validation Rule where the context is necessary to understand the functionality.   Specifically in areas with multiple outcomes and constraints.
   a. Ie: /* <Comment>   */
   b. If the logic is complex, but inline comments are not beneficial to the explanation, using the description field is an acceptable alternative
3. If you need to update an existing rule, add a date in the description and include the changes made and an “Updated By <Name>” Tag.

Profiles/Permission Sets

With Permissions on Profiles being retired in Spring ‘26, CloudKettle recommends that Permissions be Managed via Permission Set Groups.

The following will be moved to Permission Sets:

- User permissions (system and app permissions)
- Object permissions (object Create, Read, Update, and Delete [CRUD])
- Field permissions (field-level security [FLS])
- Tabs
- Record types (not defaults)
- Apps (not defaults)
- Connected app access
- Apex classes
- Visualforce pages
- Custom permissions

The following will remain on the Profile

- One-to-one relationships: Login hours/IP ranges.
- Defaults: Record types, apps.
- Page layout assignment: The future is App Builder/Dynamic Forms, so Salesforce will not invest in bringing page layout assignment to permission sets.

Descriptions should be populated by Admins to indicate the use case above.

Permission Set & Permission Set Groups

Both Permission Sets and Permission Set Groups should be Named for the Purpose they serve and with a set naming convention to make parsing permission clear.

Descriptions must be populated and explain the purpose of the Permission Set/group. An Admin familiar with security must be able to read the description and understand the purpose of the permissions and how it fits into the Permission model for the org as a whole.

For an example of some best practices around managing user permission, review this following article from Salesforce: Admin Best Practices for User Management

Best Practice Recommendations for Permissions

A permission set group should be used to make a bundle of permission that's assigned to a subset of users, a role or a person.

A permission set should be used to grant a specific level of Access to an object, set of permissions. Think of this as a building block that you would need to make up a component of multiple functional Permission Groups. Or alternatively as the most granular one off set of permissions that you need to assign to certain users.

CloudKettle recommends that Permission set Groups be created for Roles at larger organizations:

For example the PERSONA:SalesUser Permission set group contains the basic permission for that role.

If required PERSONA:SalesUser Permission set group can then be referenced in a larger permission Set Group:

PERSONA:SalesUser:AMEA which contains additional permission sets for fields and features used only by AMEA.

PERSONA:SalesUser:EU would also contain the PERSONA:SalesUser Permission Set Group as well as an additional number of permission sets for fields and features used only by AMEA.

Integration Policy

Integration User

An Integration User is a dedicated (not used by any human) full Salesforce license that has a custom Profile, Permission Set and is used for any third party integrations like marketing automation, CTIs, data enrichment tools, and even your own custom API work that ties in with your instance. Integration Users are particularly important for the tools listed above because they tend to update thousands (or tens of thousands) of records a day and have a huge impact on your instance.

In short, having an Integration User is a more secure, auditable way to move data into and out of your instance without relying on an existing user's license. Salesforce provides you with up to five free integration user Licenses depending on your orgs edition. To learn more on how to leverage these licenses, read our blog on the Integration User License.

CloudKettle recommends that every org has at least one integration user for third party integrations. For larger high volume integration such as Marketing Automation Platforms, CloudKettle recommends a separate integration user for auditability.

AppExchange/Integration Policy

CloudKettle recommends that a Steering Committee be in charge of approving integrations with Salesforce. AppExchange Apps and other third party integrations are excellent ways of effectively and efficiently adding functionality to your org, but can come with significant monetary, functional and maintenance considerations.

CloudKettle recommends that your company's procurement process be adapted and applied to Salesforce integration to include a technical analysis of the product and its effect on the architecture of the Org, as well as predicted maintenance costs.

When a solution is settled upon, ownership of the integration must be assigned to a Stakeholder as well as resources budgeted for maintenance of the integration.

Application Allow Listing

By default, any user can authorize an integration into Salesforce with the correct permissions. This can cause issues, as any user can connect to any application regardless of that application's required data permissions, security infrastructure, etc. By not restricting application access via whitelisting, organizations expose themselves to the following risks:

- Users authenticating into unapproved applications that violate security policies
- Unapproved users exporting and modifying data through applications, such as Dataloader
- Granting data access to unapproved applications that otherwise should not have access to Personally Identifiable Information

CloudKettle recommends enabling Application Allow Listing to increase the security of the org for the above reasons. By enabling Application Allow Listing, Admins control which Users and Profiles can access specific Connected Apps in Salesforce, allowing them to restrict application access to tools reviewed and approved by the organizations Admins.

Documentation and Maintenance

CloudKettle recommends that all integrations be documented and included in the Org Handbook, Data Flow Diagram and architecture diagram depending on the complexity and contents of the integration.

For larger integrations such as marketing automation, enrichment or lead routing platforms CloudKettle recommends that a regular meeting be established between the Stakeholder and the Salesforce team to review the current state of the integration as well as any pain points or changes required. The cadence of these meetings will depend on the needs of organizations, in general CloudKettle recommends a quarterly check-in.

Code Standards Documentation

Apex Classes

1. Before writing Apex code, consider if there are any viable declarative alternatives first. Requirements that do not consist of complex logic or complicated batch processing can likely be done with Flow Builder. See Record-Triggered Automation for more information.
2. Follow the official naming conventions.
3. All Apex code must be bulkified. See this article for a list of key best practices.
4. Be mindful of governor limits:
    a. No SOQL queries in for loops
    b. No DML statements in for loops
    c. Avoid heap size limits by using SOQL for loops. Only query for the fields required in your code. Help document
    d. Take into account the CPU time limit per transaction and make use of asynchronous apex whenever possible. Help document
    e. Full list of Execution Governor limits.
5. Add code comments to explain what your code does. Read the “Code comment rules” section for more details.
6. Prefer using smaller methods to larger ones. This makes the code modular and easier to read.
7. Run the PMD static code analyzer which is distributed with “Salesforce Extension Pack (Expanded)” VsCode extension. This will highlight any potential areas where best practices are not followed, including security guidelines.

Code comment rules

1. The start of each source file should include a comment with information about the class authorship, creation date, and a brief description of its purpose.
2. Each apex method should include these tokens if applicable: @author, @description, @param, @return.

3. Comments that do not fall under point 1. or 2. should express intent and provide meaningful information that is not obvious by simply reading the code. With that being said, we should still analyze if the code can be rewritten in a way that removes the need to add an additional comment.
   a. Comments are difficult to maintain. The older a comment is, the more likely it is that it conveys inaccurate information.
   b. Code evolves, new requirements are implemented, other parts are deprecated etc. Chances are comments will not be accurate if not properly maintained.
4. Unless required by the project in development, comments should not be used as a tool for tracking historical changes, otherwise the comments portion of the code becomes longer than the code it describes. Version control systems can be used to address this need.
5. Avoid writing TODO comments. Oftentimes they are forgotten or ignored. Create a technical debt ticket in an issue tracking system such as Jira.
6. The standard javadoc can be used as an example to document code.

Unit tests

1. All Apex classes must have a corresponding test class.
2. Code coverage must be as close as possible to 100%.
3. Where applicable, test the following scenarios in your unit tests:
   a. Single records
   b. Bulk operations
   c. Positive test cases
   d. Negative test cases
   e. Testing using specific users (non-admin)
4. Every test method must use the new Assert class to validate that the expected outcome does indeed match the results of running the unit test.
   a. Always include an assertion message that clearly explains why the assertion failed.
   b. A single assertion per test method should be used to test only one aspect of the functionality at a time. Exceptions to this rule are allowed when the test setup is complex, or multiple assertions are used to test the same concept.

5. Do not use the SeeAllData=true annotation. Unit tests should not rely on existing data to test functionality.
6. A Test Factory class should be used to help set up test data. This ensures that any future changes can be made directly in the factory class, reducing the risk of having to go through multiple test classes to fix how the data is set up.
7. Use Test.startTest() and Test.stopTest() to reset the governor limits before executing the code that needs to be tested.
8. Do not hardcode Ids.
9. Follow Salesforce's Testing Best practices

Resources

1. Official Apex Developer Guide
2. Sample projects: Code samples and SDKs
3. Solutions and design patterns for common use cases: Apex recipes
4. Best practices as outlined by the Success Cloud Coding Conventions Trailhead module

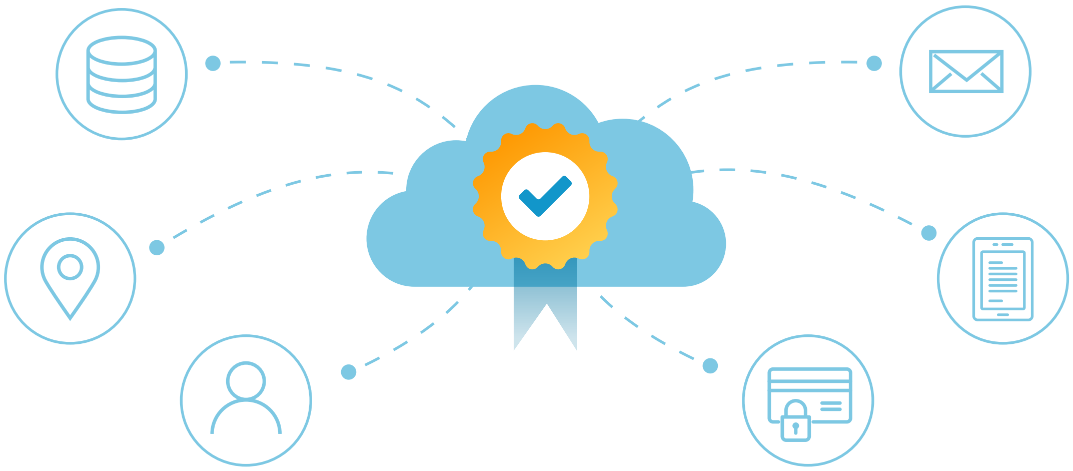
Conclusion

There are many areas of Salesforce which require oversight and governance. Setting up a Center of Excellence within your organization is one of the best ways to start developing policies that are tailored to your company's unique needs.

Release management, technical debt, data governance, metadata governance, integrations, and code standards are all topics for consideration when it comes to policies and documentation. By reviewing this handbook, your organization will be equipped with the tools to begin the creation of these important areas of governance.

Are you interested in learning how CloudKettle can help you implement a Center of Excellence or enhance your overall Salesforce governance?

Talk to us today!



CloudKettle.com

1-800-878-4756 ext 202