# Project Full Name Configuration Management Plan

This document describes how the CivicActions/Project team approaches configuration management of the Project General Services platform.

## Contents

* [Overview](#overview)
  + [Purpose](#purpose)
  + [Scope](#scope)
  + [Roles and responsibilities](#roles-and-responsibilities)
  + [Definitions](#definitions)
* [What goes into configuration management?](#what-goes-into-configuration-management)
* [Where should all this configuration go](#where-should-all-this-configuration-go)
* [How do we test these changes](#how-do-we-test-these-changes)
* [Change workflow](#change-workflow)
* [What if a configuration is changed and it is not in Configuration Management?](#Xf90fb61eaa3fa1170faedb43b7ec5f4c251d6d7)
* [Server configuration](#server-configuration)
* [Application configuration](#application-configuration)
* [GitLab contribution guidelines](#gitlab-contribution-guidelines)
* [Forking](#forking)
  + [Branching](#branching)
  + [Squashing commits](#squashing-commits)
  + [Rebase or merge](#rebase-or-merge)
  + [When should a Merge Request (MR) be created?](#X973957195c9187d647fd60105c96f122479c844)
  + [Should MRs be assigned?](#should-mrs-be-assigned)
  + [When reviewing a MR, should the change be tested locally?](#Xcded1b13065568e12e49e567ec4a59d9aab6738)

## Overview

Project employs a combination of AWS CloudFormation templates and the Ansible software and configuration provisioning engine. Using CloudFormation, CivicActions is able to create virtual machines for each step of the CI/CD pipeline.

The server software update process, AIDE-based intrusion detection, and Git management of /etc are all managed by Ansible. Drupal application updates both security and feature based, make use of a scripted deployment process. A Git repository is used to manage and record configuration in code, templates and playbooks. Peer review, automated testing and a stakeholder review on a staging server ensure that configuration updates are deployed without problems. Should a problem be discovered, rollback to a previous version is seamlessly managed by re-deploying the previous release stored in Git.

### Purpose

The purpose of this document is to identify and describe the Configuration Management (CM) process for the Project Full Name and provide CivicActions with the necessary structure to efficiently and securely manage the configuration standards for software baselines and changes to assets within the Project authorization boundary. This plan describes the processes required to ensure that the inevitable changes to Project occur within an identifiable and controlled environment.

The Project CM Plan will ensure the following requirements are met:

* Formally documented CM roles, responsibilities, and procedures;
* A configuration control board that implements procedures to ensure a security review and approval of all proposed information system changes, to include interconnections to other information systems;
* A testing process to verify proposed configuration changes prior to implementation in the operational environment; and
* A verification process to provide additional assurance that the CM process is working effectively and that changes outside the CM process are technically or procedurally not permitted.

### Scope

This Configuration Management Plan and associated processes apply to all employees, contractors and vendors that manage change or otherwise affect the operations of the Project system including but not limited to the hardware, software, facilities or information resources. The scope of the Configuration Management Plan is to establish policy and procedures to ensure that:

* The revision status of the Project Baseline can be clearly identified, accurately recorded, and provided to at any given point in time;
* The integrity of the approved Authorization and status of the Project baseline is maintained throughout all program phases/sprints;
* Coordination of approved changes are vetted in an effective and timely manner; and
* Changes to the defined Baseline is controlled and evaluated for impact on all related system aspects including security, and incorporated only after review and approval by the personnel.

### Roles and responsibilities

Project shall maintain an active Configuration Control Board (CCB) which will be established as a formal approval authority for changes. It primarily exists to control changes to the Project architecture (e.g., deployment of new software, code, or major architectural change).

The following **Roles** will be involved in configuration management activities and make up the Configuration Control Board:

* Project Program Manager
* Information Systems Security Manager (ISSM)
* CivicActions Team
  + Project Manager (PM)
  + Information Systems Security Officer (ISSO)
  + Infrastructure Support Team
  + Technology Lead (TL)
  + Development Team
* Project Managers

The **Program Manager** or a **Designated Represenatative (DR)** shall:

* ensure that appropriate roles, responsibilities, and access controls are assigned to support an effective Configuration Management Process;
* manage the Change Control Process, Change Management policy, and associated processes that are essential to the integrity of the Change Control Board;
* provide direction for Project sprints, ensuring that changes requiring modifications to the contract are submitted as required; and
* attend Configuration Control Board meetings.

The **Information Systems Security Manager (ISSM)** is the liaison between the PM and the CivicActions Team for CM-related actions, and shall:

* lead monthly Configuration Management meetings;
* ensure that CM changes are accurately assessed, documented, and disseminated to prevent any potential impact to the Project Authorization;
* analyze changes to Project to determine potential security impacts prior to change implementation;
* ensure that required stakeholders maintain active participation within the Project CCB; and
* attend Configuration Control Board meetings.

The **CivicActions Team** is responsible for the Project architecture and its components. The CivicActions Team tests and deploys Project components, modifies existing software components, and identifies potential Project enhancements. The team is composed of several roles:

* The **Project Manager (PM)** is responsible for shepherding the Agile process that is used to develop and maintain Project throughout any requested or required configuration changes.
* The **Information Systems Security Officer (ISSO)** develops and implements processes and procedures to insure the security of the Project General Service as it grows and changes through use and updates.
* The **Infrastructure Support Team** is authorized to make changes to the underlying Project infrastructure and components. This team shall ensure that a central inventory is maintained and updated as information system components are modified/added/removed to/from the Project environment.
* The **Technology Lead (TL)** manages the change process of the Project application, oversees the testing and staging operations, and is directly involved in the deployment of new releases.
* The **Development Team** is tasked with implementing newly requested features, mitigating reported bugs, and developing test systems to ensure the proper operation of the system as it undergoes changes.

The **Project Managers** are responsible for the day-to-day operation of the Project Platform system, and maintain close communication with Project users and/or organizations. This team is responsible for acting as a liaison between the Project user base and the CivicActions Team to ensure that the Project system is up and operational, and coordinating minor changes to the Project Baseline. Attends Configuration Control Board meetings as needed.

#### Project Working Group

The Project Working Group (WG) is comprised of the members of the current sprint, including at a minimum, the Program Manager or DR, a Project Manager, and a Technology Lead. The WG coordinates minor Project changes (e.g., setting changes within the Drupal application and minor operating system updates) between the CivicActions Team and Project Managers. The Project CCB delegates this authority to the Project WG to provide a more streamlined CM control mechanism for changes that do not affect the authorization of the Project system. Although the WG is less formal than the CCB, all requests and decisions must still be documented through the JIRA ticketing system.

### Definitions

The Configuration Management Process is comprised of a collection of activities focused on establishing and maintaining the integrity of the Project baseline, through control of the processes for initializing, changing, and monitoring the configurations of assets within the Project authorization boundary. This process is administered by CivicActions in collaboration with the Project Program Manager. The Program Manager, in collaboration with the ISSM shall ensure define and implement configuration baseline process and standards for:

**Configuration Item (CI)** - An identifiable part of a system (e.g., hardware, software, firmware, documentation, or a combination thereof) that is a discrete target of configuration control processes.

**Baseline configuration** - A set of specifications for a system, or CI within a system, that has been formally reviewed and agreed on at a given point in time, and which can be changed only through change control procedures. The baseline configuration is used as a basis for future builds, releases, and/or changes.

**Configuration Management Plan (CM Plan)** - A comprehensive description of the roles, responsibilities, policies, and procedures that apply when managing the configuration of products and systems. The basic parts of a CM Plan include:

* **Configuration Item Identification** – Methodology for selecting and naming configuration items that need to be placed under CM.
* **Configuration Change Control** – Process for managing updates to the baseline configurations for the configuration items.

Other Commonly used terms used within the Configuration Management Process include:

**Baseline** - A current and comprehensive baseline inventory of all components required to support Project operations; these components are part of the System Inventory and can be changed only through formal change control procedures. The baseline includes sufficient detail to re-create the Project General Service. Baselines exist for Software and Infrastructure, and redundant copies of the Baseline are stored by CivicActions in a location separate from the Information System.

**Baseline Change Request (BCR)** - A formal written request to initiate a change to a baseline document.

**Configuration Control Board (CCB)** - A review panel that evaluates and/or approves changes to the Project baseline.

**Code Commit** - A definitive change to any source code that defines the Project software, or Project virtual infrastructure, or other supporting Project asset or document which contributes to the Project Information System. Each code commit is assigned a unique ID, and all code commits are part of a permanent record. All changes to Infrastructure and Software Baselines are executed through code commits. It should be noted that not all code commits result in changes to Baselines.

**Version** or **Release** - (1) A uniquely identified snapshot of a build that represents some identifiable milestone of functions and capabilities of the Information System; or (2) a uniquely identified snapshot of a document representing some identifiable milestone of content.

## What goes into configuration management?

In short, everything needed to run and operate the platform that is not a *secret*. (*tbd: secret key management*)

Here are some examples that are in configuration management:

* CI/CD pipeline
* Infrastructure/network configuration (CloudFormation and Ansible)
* VM setup and quantity (CloudFormation and Ansible)
* Server software configuration (Ansible)
* CivicActions-developed code (Git)
* Application configuration (Drupal features in Git)

## Where should all this configuration go

All configuration must be stored in GitLab using the following “Change Workflow” unless it is a *secret*.

## How do we test these changes

If possible, changes should be tested locally first. If local testing is successful, upload the changes to a development environment for manual or automated testing.

Security tests need to be executed in the development environment where changes are applied.

## Change workflow

1. All configuration changes must flow through a Git repository, centrally managed through GitLab, unless they contain sensitive information.
2. A change is initiated and discussed as a “Backlog” JIRA ticket in the [JIRA ticket management system](https://project.atlassian.net/secure/)
3. During Sprint Planning, the ticket is prioritized and may get moved from “Backlog” to “ToDo”.
4. The ticket moves from “ToDo” to “In Progress” when it is assigned to a developer.
5. During development, code commits are checked for style and security using githooks.
6. After development and local testing, the developer initiates a “Merge Request” (MR).
7. The MR is reviewed by someone other than the committer. Pairing via screen-sharing is encouraged and qualifies as a review. Review should include assessment of architectural design, DRY principles, security and code quality.
8. The reviewer merges the MR.
9. A continuous integration (CI) server handles automated tests and continuous deployment (CD) of the merged changes.

* All changes are deployed to a newly created test environment.
* Any and all automated tests are run.
* If all tests pass, changes can be promoted for deployment to production in the pipeline.

1. The CI/CD tool uses GitLab repositories as the single source of truth for what the platform should look like. If there are manual changes, the CI/CD tool resets the state of all systems to match.

## What if a configuration is changed and it is not in Configuration Management?

If possible, Configuration Management tools should always roll back to a known state. Other than that, these tools need to be able to “recreate” all settings from known configurations.

## Server configuration

Server configuration is handled via CloudFormation templates and Ansible playbooks and managed using Git. Once a change has been committed and pushed to the Git repository, a merge request is created. Creating the merge request triggers the CI/CD build pipeline which contains the following phases:

* **Deploy infrastructure:** In this phase the containers are created using the CloudFormation templates and Ansible playbooks.
* **Deploy services:** Services defined in the CloudFormation template are deployed. The services include the bastion host, the Drupal application, the Ilias CMS, Solr searching, and the Amazon RDS databases.
* **Validate platform:** During the validation phase, the server configuation is tested for drift detection in order to catch configuation settings that have deviated from the baseline configuation, as well as checks to determine if the applications are up and running and accessible. Nmap, OpenSCAP and Zap scans are also performed during this phase.
* **Post validation:** During the post validation, the hardened Amazon Machine Image (AMI) is checked to see if there are any updates available and, if so, they are installed.

## Application configuration

Configuration management in Drupal is handled using Drupal’s Configuration Management and hook\_update\_N modules to make the necessary changes to site configuration.

Each site has its own site\_deploy module that orchestrates deployments.

When code is deployed to sandbox, development, staging or production environments, drush config:import, which imports changes to configuration, is run, and drush updatedb is run, which runs any new hook\_update\_N functions.

For many of the common configuration tasks, Hook Update Deploy Tools methods make sure that all hook\_update\_N modules follow this model::

* make the change,
* verify it was made, and then
* report that it was made.

Records of these events are output to the terminal of the engineer deploying the code, and to Drupal Watchdog.

## GitLab contribution guidelines

Project is built and maintained by CivicActions, and the CivicActions Team follows standard code development guidelines.

## Forking

Forking is a method that can be used to modify the code base.

The CivicActions Team maintains the Project code base in a GitLab repository. The *master* is the most current version that has been deployed to production. When starting a new project, the CivicActions Team makes a copy of the *master* in GitLab; the copy, called a *fork*, is where project-specific code changes shall be maintained going forward.

Code changes are implemented using the following workflow:

1. The CivicActions Team uses the *master* to create a *fork* for project-specific code changes.
2. During the project, individual team members create branches, and then work in those branches until the code changes are ready to be committed to the fork.
3. When code changes are ready to be integrated from a team member branch into the fork, a team member creates a merge request in GitLab. (All code changes are implemented using merge requests.)
4. Another team member reviews the merge request, performs a code review, and approves the merge request so that the code changes can be integrated into the fork.

### Branching

Branching is another method that can be used to modify the code base.

Each code repository has at least two branches:

1. **Master branch.** The master is used for development. The CivicActions Team can rebase and create merge requests in the master branch.
2. **Production branch.** The production branch is deployed to the production server. When deploying changes to production, the release manager copies code from the master branch to the production branch.

The CivicActions Team might create a branch from the upstream repository when multiple developers need to collaborate on something that cannot be continuously merged into the master branch. The rationale for branching within a team is that paired collaboration on a single branch avoids certain types of friction:

* Having to process merge requests from multiple forks in order to integrate changes to the upstream branch
* Having to add team members to forks as *collaborators* so that they can contribute in short-lived forks

When team members contribute directly in a branch, CivicActions can modify work-in-progress (WIP) merge requests and encourage collaboration across the Cloud Operations team.

### Squashing commits

[Squashing commits](https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History#Squashing-Commits) is allowed and encouraged within an engineer’s branch, but discouraged, except in rare instances in master and production branches which are fast forward only and block force pushes.

### Rebase or merge

The team prefers [rebasing over merging](https://www.atlassian.com/git/tutorials/merging-vs-rebasing/). Ongoing work should always be rebased upon the master branch.

### When should a Merge Request (MR) be created?

Work-in-progress MRs are encouraged. If you create a work-in-progress MR, you might also make it plain in the MR name with a [WIP] prefix. When a MR is ready for review, remove the [WIP] label. A MR with a WIP label is blocked from merging by GitLab.

Merge requests should be created whenever code is ready for review, prior to being merged into the master branch.

### Should MRs be assigned?

MRs are typically not assigned in GitLab, unless someone specifically needs to sign off on the change.

You can request a review using GitLab’s built-in tools, mention someone in the MR with the @ notation, or contact them outside the GitLab context to request a review.

### When reviewing a MR, should the change be tested locally?

Whenever possible, the proposed changes should be tested locally. Because of the nature of many of the Project repositories and deployment environments, local testing is not always possible or practical. Visual code review, however, is always required. In the event that merged code breaks the dev environment, the decision will be made at the time whether to revert the merge.