



The content described herein is intended to outline our general product direction for informational purposes only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described herein remain at the sole discretion of Atlassian and is subject to change.

# DevOps best practices to better manage your Jira instances



# Hello there!

.....



Gil Hoffer  
Co-Founder & CTO  
Salto



## Investors



# SaaS apps run the modern company



**All these apps  
require deep  
customizations**

# Challenges (1/2)

.....

1

How do we really know what is implemented?

2

Why a change was made, by whom, and when?

3

What will be the impact of a planned change? Inside Jira? On other systems?

4

How do we develop changes reliably, without impacting production?



# Challenges (2/2)

.....

5

How do we collaboratively review those changes in a team and create collective code ownership?

6

How do we reliably promote those changes to production?

9

How do we know what is already broken in our implementation (broken automations, screens, workflows, etc.)

7

How do we roll back, if needed?

8

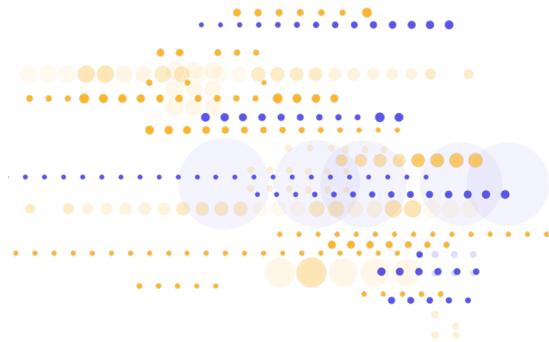
How do we keep a paper trail for compliance?



# The “as code” concept completely transformed IT

A fusion between development and operations helped IT take a big leap forward

- ❌ Manual error-prone processes
- ❌ Organizational silos
- ❌ Long delivery cycles
- ❌ Infrequent releases
- ❌ Reduced quality



- ✅ Faster release cycles
- ✅ Shorter delivery sessions
- ✅ Increased quality
- ✅ Continuous delivery

Automating operational tasks with code

# Why does “as code” matter (subset)?

.....

Version-controlled and immutable, makes auditing easier and rollbacks feasible

Linters and static analysis tools to enforce consistency



Maintainable, testable, and collaborative – reduces the bus factor

Modular, composable, and separate

Predictable, repeatable, and consistent

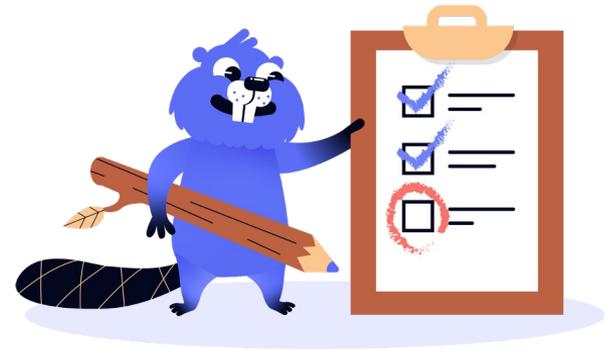
# Business engineering

We should adapt best practices, methodologies, and tools from the Dev and DevOps world into how we manage our business applications.



# Business Engineering 101 (1/2)

- Having a clear and **complete textual representation of the implementation** (declarative and code) will enable visibility and easily answer “what is implemented?” (As easy as ⌘+F)
- **Version** that representation **in a source control system** (such as Git) and be able to answer questions on changes over time.
- Code analysis tools can **answer** questions on **dependencies** and **change impact**.
- Create **shared responsibility** within the **team** — everyone should be aware of all major changes.



# Business Engineering 101 (2/2)

- **Always** develop in high-fidelity **sandboxes**.
  - Not impacting production
  - Not testing on nonrepresentative setup
- A change proposal → **feature branch in Git and pull request**.
- **Promoting a change** from dev to UAT to prod → **git merge and deploy**.
  - The holy grail – **CI/CD**
  - The holier grail – trunk-based development (TBD)
- **Rollback** → Git revert and deploy.
- **Maintain** a proper **paper trail** for changes → tie change requests (for example, from Jira) to Git commits.



# Cross-business application applicability

- Everything we discussed so far is **not** Jira-specific.
- As an industry, we should have a standard way to practice business engineering **across** our entire stack.



ORACLE®  
NETSUITE

workato Jira

zendesk

stripe

zuora

# Examples



# Represent everything “as code”

Search for elements, files and text

Elements Files Content

- ▼ Jira
  - ▼ Records
    - Application Property
    - Application Role
    - Automation
    - Board
    - Dashboard
    - Dashboard Gadget
    - Field
    - Field Configuration
    - Field Configuration Scheme
    - Filter
    - Group
    - Issue Event
    - Issue Link Type
    - Issue Type
    - Issue Type Scheme
    - Issue Type Screen Scheme
    - Notification Scheme
    - Permission Scheme
    - Priority
    - Project
    - Project Component
    - Project Role
    - Project Type

```
Align_Hour.nacl x
1  jira.Board Align_Hour@s {
2      name = "Align Hour"
3      type = "kanban"
4      admins = {
5  >      users = [ ...
10     ]
11     }
12     location = {
13         projectId = jira.Project.instance.Salto_SaaS@s
14     }
15     filterId = jira.Filter.instance.Filter_for_Align_Hour@s
16     columnConfig = {
17         columns = [
18             {
19                 name = "To Do"
20                 statuses = [
21                     jira.Status.instance.Pending,
22                     jira.Status.instance.Design,
23                     jira.Status.instance.To_Do@s,
24                     jira.Status.instance.PM,
25                 ]
26             },
27             {
28                 name = "In Progress"
29 >             statuses = [ ...
33         ]
34         },
35         {
36             name = "Done"
37 >             statuses = [ ...
42         ]
43         },
44     ]
45     constraintType = "issueCount"
46 }
47 subQuery = "fixVersion in unreleasedVersions() OR fixVersion is EMPTY"
48 }
49
```

# Render the metadata in a friendly way (1/2)

The screenshot displays the Salto configuration interface. On the left, a navigation pane under 'Elements' shows a tree structure for 'Jira' > 'Records' > 'Board'. The 'Align Hour' element is selected and highlighted in blue. The main content area shows the configuration for 'Align Hour' (INSTANCE) under 'jira APP > Board TYPE'. The configuration is organized into sections: 'VALUES', 'References', and 'DEPENDS ON'. The 'VALUES' section includes fields for Filter ID, Name, Sub Query, and Type. The 'References' section lists 'Filter', 'Status', and 'Project'. The 'DEPENDS ON' section lists 'Filter', 'Status', and 'Project'. A footer note suggests using 'Content search' to find additional references.

**Elements** Files Content

- ▼ Jira
  - Records
    - Application Property
    - Application Role
    - Automation
    - Board
      - Align Hour**
      - Anastasia's Board
      - Avi's Board
      - BILLING2 board
      - BIZAPP board
      - Change Management
      - DOC board
      - Data and Instrumentation
      - Deploy
      - ECS board
      - Emily's Board
      - Engineering
      - Fun board
      - Jira
      - LANG board
      - Marketing Operations
      - Miki's board
      - Miriam board
      - My Kanban
      - NA board
      - S&I R17 board

**Align Hour** INSTANCE

jira APP > Board TYPE

▼ VALUES

Filter ID	Filter for Align Hour (Filter)
Name	Align Hour
Sub Query	fixVersion in unreleasedVersions() OR fixVersion is EMPTY
Type	kanban

▶ Admins

▶ Column Config

▶ Location

References

▼ DEPENDS ON | 13

- ▶ Filter | 1
- ▶ Status | 11
- ▶ Project | 1

Try [Content search](#) to find additional references across your entire configuration data

# Render the metadata in a friendly way (2/2)

The screenshot displays a web application interface with two main sections:

- Left Panel (File Explorer):** Titled "Elements", it shows a hierarchical tree structure. The "Board" folder is expanded, and "Align Hour" is selected and highlighted with a blue box. Other boards listed include Anastasia's Board, Avi's Board, BILLING2 board, BIZAPP board, Change Management, DOC board, Data and Instrumentation, Deploy, ECS board, Emily's Board, Engineering, Fun board, Jira, LANG board, Marketing Operations, Miki's board, Miriam board, My Kanban, NA board, and S&I RI7 board.
- Right Panel (Configuration):** Titled "Align Hour INSTANCE", it shows the configuration for the selected board. The breadcrumb "jira APP > Board TYPE" is visible. The "Column Config" section is expanded, showing:
  - Constraint Type:** issueCount
  - Columns:** A list of columns with their names and statuses:
    - 0: Pending (Status)
    - 1: Design (Status)
    - 2: To Do (Status)
    - 3: PM (Status)
  - Next Column:** Name: In Progress

# Understand dependencies

**Elements** Files Content

- Jira
  - Records
    - Application Property
    - Application Role
    - Automation
    - Board
    - Dashboard
    - Dashboard Gadget
    - Field
      - Affected Services
      - Affects Versions
      - Approvers
        - Contexts
        - Approvers**
      - Assignee
      - Attachment
      - Category
      - Category Jwm Category
      - Change Completion Date
      - Change Reason
      - Change Requires Downtime
      - Change Risk
      - Change Start Date
      - Change Type
      - Comment
      - Compass
      - Components

## Approvers INSTANCE

jira APP > Field TYPE

**VALUES**

Description	Contains users needed for approval. This custom field was created by Jira Service Desk.
Name	Approvers
Type	com.atlassian.jira.plugin.system.customfieldtypes:multiuserpicker
Contexts	Approvers__c_Default Configuration Scheme for Approvers (Custom Field Context)

**References**

**USED BY** | 7

- Field Configuration Item | 3
- Screen | 4 ...

**CHILDREN** | 1

- Custom Field Context | 1

**DEPENDS ON** | 1

- Custom Field Context | 1

# Monitor configuration changes

🔔 Notify on automation changes Cancel Save

On any change ↕ Switch to simple selection

jira.Automation.instance.\* 48

+ Filter by references

Notify me via

Slack ▼ #gil-tracker-notifications ▼

+  Notify even if there are no new changes

🗑️ Delete notification Cancel Save

🔔 Notify on new workflows and workflow schemes 🔴 Edit

[+ Add another notification](#)

# Compare environments

### Promote from UAT to Prod

Deploying from `jira-uat` to `jira-production`

#### Element Selection

5775 Added | 85 Modified | 726 Deleted | 6586 All Char

Name ↓

- Project (4)
  - Biz Apps
  - Salto
  - Salto SaaS
  - Website
- ProjectComponent (15)
  - Salto SaaS@s Tracker
  - Salto SaaS@s UI
  - Salto\_\_Jira Adapter
  - Salto\_\_Netsuite Adapter

### Salto\_\_Jira Adapter INSTANCE

jira APP > ProjectComponent TYPE

2 changes (0/2 selected) | 1 of 2

[Element](#) [Changes](#) [NaCl](#)

▼ VALUES

<input type="checkbox"/> Assignee Type	<b>COMPONENT_LEAD</b> → <b>PROJECT_DEFAULT</b>
Name	Jira Adapter
<input type="checkbox"/> ▶ Lead Account ID	

# Version control

salto-io/jira\_audit\_gil MAIN

4 unpushed files [Preview Push](#)

### Commit History

Today

- Gil Hoffer `D942CC1` 4:03 pm  
BIZAPP-461 Add automations for project GR
- Gil Hoffer `528D4DD` 4:02 pm  
BIZAPP-460 implement project GR

02/03/2022

- Gil Hoffer `5DD7F10` 7:01 pm  
baseline
- Gil Hoffer `B1C2B52` 6:47 pm  
Salto initial commit

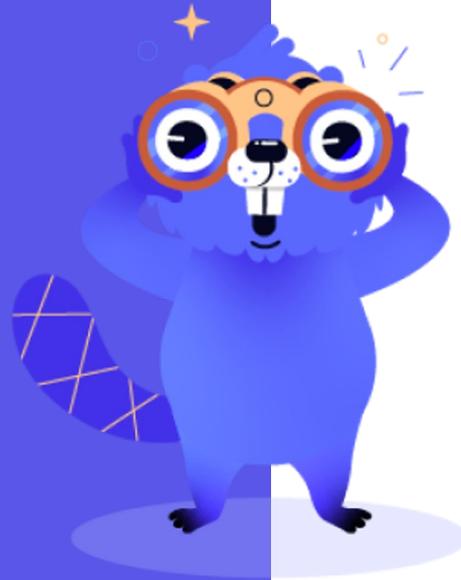
[View commit on Github](#)

# Deploy changes (from UAT ☒ Prod)

The screenshot displays the Salto Jira production deployment interface. At the top, the environment is identified as 'jira-production' with navigation tabs for 'EXPLORE', 'AUDIT', 'COMPARE & DEPLOY', 'MONITOR', and 'SETTINGS'. A 'Preview Deployment' button is visible. The deployment is labeled 'Deployment 456' and shows it is deploying from 'jira-uat' to 'jira-production'. Below this is the 'Element Selection' section, which includes a search bar and summary statistics: 6809 All Changes, 5994 Added, 91 Modified, 724 Deleted, 33 Selected, and 8 Missing Dependencies. A table lists the selected elements with their names, types, change status, required dependencies, and additional dependencies.

Name	Type	Changes	Required Dependencies	Additional Dependencies
Records (33)			49/59	35/783
Automation (1)			1/1	
Alert on high incident severity	Automation	+	1/1	0/0
Field (6)			4/7	8/146
FieldConfiguration (9)			12/12	15/488
FieldConfigurationScheme (1)			3/3	1/4
IssueTypeScreenScheme (1)			1/1	1/2
PermissionScheme (1)			1/1	1/4
Alon Permission Scheme	PermissionScheme	+	1/1	1/4
Project (1)			15/15	1/54

# Recap



Manage your customizations “as code.”  
Apply DevOps best practices to the way you manage them.  
You can start **today**, even by taking small steps.

# Stay in touch



salto.io



@salto-io



@salto\_io



Free tier (free for life)