The UNOFFICIAL BIG Book of Atlassian

By Alex Ortiz and Bernandina Ortiz
Published by APETECH LLC

appfire

The Appfire advantage

- Innovative, purpose-built products
- Powerful customer support portal
- One EULA for all apps
- Best-in-class software vendor
- Enterprise-grade security & privacy

visit appfire.com

Disclaimer

Please note that this book, "The Unofficial Big Book of Atlassian", is an independent publication authored by APETECH LLC. It is not created by, affiliated with, endorsed by, or in any way associated with Atlassian Corporation Plc or any of its subsidiaries or products. All trademarks and registered trademarks mentioned within this book are the property of their respective owners.

The information, insights, and guidance provided in this book are intended for general reference and informational purposes only. While every effort has been made to ensure the accuracy and utility of the content, the dynamic nature of business, technology, and individual circumstances means that applying the concepts and suggestions herein may not guarantee specific outcomes or solutions for all your company, team, or personal challenges. This book is designed to serve as a guide and a source of potential strategies, but it should not be considered a definitive or comprehensive solution for every unique situation. Readers are encouraged to exercise their own judgment and discretion when implementing any advice or methodologies discussed.

Please also be aware: Atlassian Corporation Plc and its associated entities continuously update and enhance their applications. As such, some guides or instructions within this book might not immediately or appropriately reflect those changes following an Atlassian application update. We recommend consulting official Atlassian documentation for the most current information.

All content within this book, unless otherwise noted, is the property of APETECH LLC. However, it is specifically stated that the coloring pages featured in this book are the sole ownership of Kythera Contreras. Furthermore, any advertisements included within this book are the property of the respective companies that submitted them.

Finally, please forgive any typographical errors you may encounter within these pages. They serve as a gentle reminder that this work was thoughtfully crafted by human hands, not by artificial intelligence.

About the Authors

Alex Ortiz
APETECH LLC CEO

Alex started making YouTube videos back in 2021 to document his journey of mastering Jira. Little did he know the videos would help hundreds of thousands of people. In 2024, Alex had the opportunity to partner with Atlassian and create a series of training videos for all types of Atlassian customers. Two years later, the idea of creating a conference where you could attend the trainings in person was born. Alex Co-created the Atlassian Builders' Summit as a conference for the community by the community. Whenever Alex isn't busy creating videos and tutorials, he's helping people all over the world with their Jira problems.

Bernandina (Lina) Ortiz
APETECH LLC CTO

Lina is an embedded software engineer, editor, and creative problem solver with a passion for programming, AI, and simplifying complex ideas. Lina plays a key role behind the scenes, organizing, designing, and refining the details that bring big ideas to life. Her thoughtful approach and "get it done" mindset have shaped everything from video productions to large scale events, including the coordination and preparation that made this book possible. Driven by curiosity and a love of learning, Lina continues to explore the intersection of technology, creativity, and education, always finding new ways to make knowledge accessible and inspiring.

About APETECH

APETECH LLC is a family-owned business with a simple mission: to help people all over the world get the most out of their tools and processes. We believe that when people are empowered with the right knowledge, they can do their best work with confidence.

This book was created as a friendly, go-to guide for anyone using Atlassian tools. Inside, you will find practical tips, step-by-step tactics, and big-picture strategies to help you become more effective, whether you are an Admin, Project Manager, or Forge developer. Think of it as a resource you can always come back to whenever you need clarity, new ideas, or just a little extra help.

APETECH LLC is also the parent company of the YouTube channel *Apetech Tech Tutorials*, where Alex Ortiz shares free resources, tutorials, and insights to support the global Atlassian community. Beyond tutorials, we collaborate with Atlassian Marketplace partners to provide honest and thoughtful showcases of apps designed to solve real-world problems.

At its core, APETECH LLC is run by husband-and-wife team Alex and Lina, with the joyful help of their children. Together, we bring not only expertise but also heart into everything we do, because for us, it is about more than tools, it is about helping people succeed.

Special Thanks

I'd like to personally thank Shawn Doyle for being a great mentor, partner, and friend during this entire journey. His years of experience have helped shaped this event and this book. We've had countless conversations on how to make the event better, how to make this book better, how to provide a crazy amount of value in just a couple hundred pages. Shawn's company ReleaseTEAM, Inc. was the first team to jump on and support the development of this event and this book and were the first to offer speakers for the sessions.

Thanks Shawn and the rest of the folks over at ReleaseTeam!!

-- Alex Ortiz

Thanks to My Boys

Creating something this big is not easy, especially when it's your first time. I want to thank my boys, Alex, Jeremy, and Isaac, for always being there for me. Nothing is possible without a great support system behind you. Jeremy, thank you for your amazing hugs and for helping me around the house when I felt overwhelmed with all the to-dos. Isaac, thank you for always making me laugh with your silly videos that helped me through the tough days. And Alex, thank you for your support and for bringing me along on this journey. Most of all, I thank God for His great love and grace over my family, for the blessings He continues to bring us, and for giving me the strength to balance all the roles I play as a wife, mother, human, and engineer.

--Bernandina Ortiz

Message to our Sponsors and Attendees

We are beyond grateful to all of our amazing sponsors and attendees who made the Atlassian Builders' Summit possible. Your support, energy, and participation turned this vision into a reality. To our sponsors, thank you for believing in the mission and helping us create a space where builders can connect, learn, and grow. To our attendees, thank you for showing up with open minds and open hearts, ready to share your knowledge and experiences. This event wouldn't have been the same without your enthusiasm and contributions. Together, we've built something truly special, and we can't wait to continue this journey with you.

--Alex and Lina

Atlassian Builder's Summit is teaming up with Nurturing Purpose and Labdoo to build new futures by upcycling unused working devices.

Give a New Life to Your "Dootronic"

At Labdoo, uses the word dootronics (as in labdoo electronics) to refer to any computer device that can be used for education purposes. Contact Nurturing Purpose to coordinate the donation of your unused electronic devices. Let's build futures together.

About Labdoo

Labdoo is a global collaborative social network of grassroots volunteers using CO2-neutral means to provide schools in need with unused computers loaded with powerful educational software

Edoovillages

With this effort we are supporting these schools supported by Labdoo that receive refurbished laptops with educational software, helping bridge the digital divide in underserved communities

Nepal
Familia de Hetauda

Argentina
FUNASER

India
Pune FamilyResource Center

Moshi, Tanzania
DelightNiche bootcamps
MOSKISA Kids of Street soccer academy
Mikono Salama school

Atitlan,Guatemala
David LaMotte community school EDEE - Special Education school

Doortrips

Travel with purpose—make your trip a Dootrip! Carry a laptop, deliver education, and help bridge the digital divide. Contact us if you're traveling soon.

About Nurturing Purpose

Nurturing Purpose empowers communities to grow by supporting local leaders, raising awareness, and mobilizing resources. Our focus is on building strong foundations so communities can thrive independently and become tomorrow's changemakers.

nurturingpurpose.org

Thank You to Our Sponsors

Presenting Partner

gryd.io

Summit Sponsor

appfire

Technical Tracks Sponsors

revyz

clearpath by moserlabs

APWIDE

Gallerie Sponsor

catapult LABS

Innovation Sponsor

Multiplier

Supporting Sponsors

RT Release Team DevOps Tool Specialists

re:solution

APPsolute Best

CollabSoft VENTURES

PREPEND

GitProtect.io by Xopero ONE

GETINT

Service ROCKET

YASOON

catworkx

BLUE RIDGE .cx

perforce Gliffy

TJL THE JIRA LIFE

APETECH

draw.io

Aura Apps

STILTSOFT

Trundl

Creative Partner

Kythera Contreras

Table of contents

Atlassian Administration

GETTING STARTED WITH JIRA ADMINISTRATION

Jira	**Done**	**Sprint**	**Field Configuration**	**Bug**
Assignee	**Permission Scheme**	**Velocity**	**Epic**	**Feature**
Work Item	**Administrators**	**WIP Limit**	**Story**	**Initiative**
In Progress	**Scrum Board**	**JQL**	**Task**	**Due Date**
To Do	**Kanban Board**	**Screen Scheme**	**Subtask**	**Priority**

RELEASETEAM

DevOps Tool Specialists

License Sales & Management

Better License Management Starts Here – And Costs Less

releaseteam.com

Consulting & Support

Experienced Atlassian Experts Deliver Solutions

Staff Augmentation & Mentoring

Scale your team with the right partner

Trusted By

- Public Sector
- Private Sector
- All Industries
- U.S. & Canada

Colorado Secretary of State
mosaicx
OSHKOSH
Trimble
Raytheon Technologies
WISCONSIN
TEXAS A&M UNIVERSITY
ZeniMax MEDIA INC.
copic

www.releaseteam.com
info@releaseteam.com
866-887-0489

Veteran Owned Business VOB VERIFIED PROUD MEMBER

youtube.com/@releaseteam

Determining a Jira Space Template Guide

Jira ships with many different space templates to choose from. You want to pick the space template type that best fits your specific needs. The first thing you need to evaluate is which "Jira" space are you intending to use. Atlassian presents you with all of the different "Jira" products available which makes choosing a template a little harder. These are the different "Jira" products you can choose from:

Jira (Formerly known as Jira Software) - Project Management
Jira Service Management (Assuming you have JSM) - Intake / Helpdesk
Jira Product Discovery (Assuming you have JPD) - Ideation / Roadmap

For each of these 3 products, you then have a plethora of different templates to choose from.
Make sure you first determine which product you intend to use, before picking a template.

Made for you	Atlassian makes recommendations based on templates that you have recently used in the past.
Bundles	Atlassian allows you to quickly set up multiple spaces at the same time to save you time. There are a couple of out of the box options or you have the ability to make your bundle if you find yourself making spaces frequently.
Custom templates ENTERPRISE	If you are an enterprise customer, you can create repeatable templates that share Boards, Work Types, Fields, Workflows, etc. Very similar to setting up the "perfect" space and then using that space as the basis for future spaces. This is just more formal and only for Enterprise subscribers.
Software development	When you need a Scrum board, this is the only space type you should select. This is typically for "agile" or "scrum" teams
Service management	For teams wanting to use JSM. You need to have purchased JSM. There are many different types of templates here that allow you to create JSM spaces. I would recommend you use the General Service Management template for a basic JSM space, or if your on the Premium version of Jira and need to have a full ITSM solution, then pick the IT Service Management Template
Work management	For teams wanting wanting to use Jira, but don't want to worry about "Agile or Scrum". Atlassian offers many different templates based on specific use cases, but I prefer starting with a blank space as your company's process will be custom enough that you'll end up modifying any of the out of the box solutions.
Product management	For teams wanting to use JPD. You need to have JPD Purchased. There are a few different templates here, but you just need a simple Product Discovery template to get started.
Marketing	The rest of the options available (Marketing, Human Resources, Finance, Design, Personal, Operations, Legal, Sales, Analytics, IT, Facilities, Nonprofit are extensions of the aforementioned templates. They just offer more specific use cases and preconfigured templates. I would recommend you check them all out and I would also recommend you personally create one of each space template in a SANDBOX environment so that you can get a feel for what they provide. I find that most teams end up modifying these templates because their processes don't necessarily match with what Atlassian offers. Because of this, for any of these options, I recommend you just go with the Basic Work Management template and customize from there. I find that it's easier to add configurations such as work item types, fields, workflows, as opposed to removing or renaming what Atlassian gives you when you pick these templates.
Human resources	
Finance	
Design	
Personal	
Operations	
Legal	

How to determine what questions to ask stakeholders before creating a Jira space

Now that you know the different space types and templates available, the most important question is how do you determine which template you need.

The first step is making sure you select the appropriate product. Jira, Jira Service Management, and Jira Product Discovery are very different products, so that step should be easy. Since this section of the book is all about Jira, let's take a look at determine which Jira template is right for you and your team.

Next, determine if the team you are creating the template will be doing agile based software development or just space management.

- If the team is doing agile based software development / agile project management, then using a Jira Software template is the only option for you as that will be the only space type that enables things like Sprints, Story Points, etc.
- If the team is not doing any agile based software development or agile project management, then you can use any of the Work Management space template types. Please note that the Software space types will still work and the Kanban style space template is a great option for teams that still want the full power of Jira, but don't want to do Scrum.

Because the Software space template types is the only template type that is going to give you the full power of Jira, let's assume that your team wants a Software template. Now you have to pick between Kanban or Scrum.

Scrum: This is the only template you can select if your team requires a proper Backlog, a Sprint, the ability to estimate work with Story Points. This is also the only template that will give you Scrum reports such as a Burndown Chart, Burnup Chart, and a Sprint Report. This template requires teams to follow the Scrum methodology which requires the team to follow the rules that Scrum defines. Atlassian also predefines the workflow for a Scrum based space template to be:

- To Do
- In Progress
- Done

Kanban: This template is very similar to the Scrum template, but doesn't provide the ability to run Sprints or visualize your Story Point estimates. The Kanban space template is a little more flexible and allows you to define a throughput method where your team plans their work and then works on items that are selected to be worked on. The Kanban template also has the option to enable a dedicated backlog, but this is not configured by default. The Kanban template is also closer to the Work Management templates but there is one key benefit.

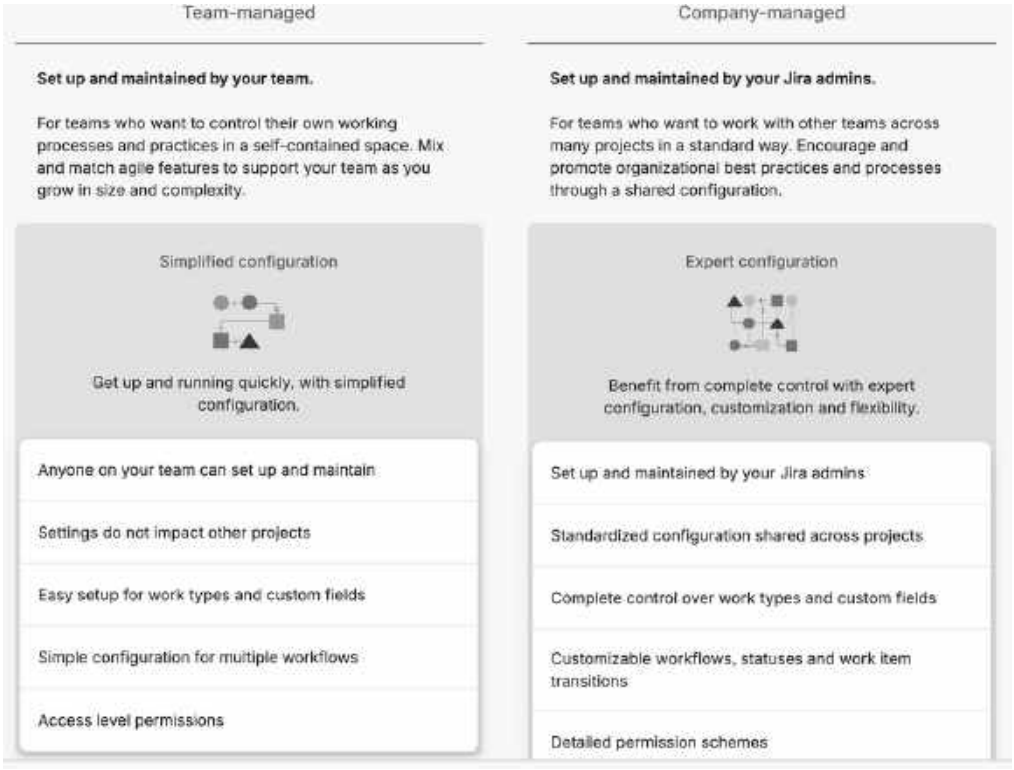
- Kanban space templates can have multiple Kanban boards (or any combination of Kanban or Scrum), while Work Management templates can only have a single board.

The Kanban space template also contains a different set of reports that favor throughput such as Cycle Time and a Cumulative Flow Diagram.
Atlassian predefines the workflow for a Kanban based space template to be:

- Backlog
- Selected for Development
- In Progress
- Done

Team-Managed VS Company-Managed Guide

Life's biggest decision you'll make in Jira. Do you pick Team-Managed or do you pick Company-Managed? Selecting the wrong template can handicap your team and the impact to your team can be severe. Use this guide to help you pick the appropriate type.



Team-Managed spaces

Team-managed spaces are great for teams that do not have a lot of Jira Administration experience. This space type allows users to get started with Jira in minutes and space configurations are contained to the space that is being configured. This is great for folks that don't have a lot of Jira Administration training as their actions will not impact the other Jira spaces. This sounds great right? Except, that all good things come at a cost. Because of the isolation, these spaces don't scale well for large teams. The concept of "sharing" configurations doesn't exist for team-managed spaces. Additionally, while work items can be linked together, the parent of any given work item must be within the same space as the child. This just doesn't scale well for larger teams where work is happening in multiple Jira spaces.

Company-Managed spaces

Company-managed spaces give you all the features of Jira. You can share configurations, fields, work. But there is a pretty big drawback. Because of the complexity of how company-managed spaces work, you really do need to know how to be a good Jira Administrator. Not understanding the impact you can have on surrounding spaces is an easy way to really break things in Jira. Luckily, there are many certifications available to help you be a great Jira Administrator. Additionally, company-managed spaces can take days, weeks, or months to fully configure because there are many different options out there that can be tweaked. The good thing is that company-managed spaces can be tailored to match your work process more accurately. The bad news is that you need to know Jira more intimately. The biggest takeaway, if you want your teams to fully collaborate together, using all the power and features of Jira, then company-managed spaces is the only type you should be using.

Setting Up A Team-Managed Space Guide

Step 1 – Create the space

- On left hand side vertical menu go to spaces and click + (Create space). As long as your Jira administrator hasn't disabled this feature, any logged in user should be able to create a team-managed space.
- Select the space template you need (Software Development for Agile project management / Software development).
- Select Kanban or Scrum (Scrum if you need to have a Sprint)
- Click on Use Template (This just shows you the features of a Scrum space template).
- Click on Select a team-managed space (The PURPLE button)
- Enter space name and key
- Select the space permissions:
  - Open (Anyone can see and edit the work in the space)
  - Limited (Anyone can see the space, but only specified user can edit the work)
  - Private (Only specified users can access and edit the work)
- Click on Next
- Skip the step where you invite users because this will add licenses ($$) to your Jira.
- For now, skip the Code and Space Pages section. Just click Continue at the bottom.

Step 2 – Configure Space Settings

- Click on the three dots (...) next to space name → Space Settings.
- Under Work types, Add/Edit/Remove work Types.
- For each Work type, Add/Edit/Remove any custom fields.
- For each Work Type, Add/Edit/Customize Workflows
- Optionally, add any work type level security (Who can create/edit specific work items)

Step 3 – Board Setup

- Configure board columns and statuses (Column name doesn't have to match Status name)
- Add any custom filters if needed
- Configure the card cover images if needed
- Configure the timeline view (schedule work by sprint or independently).

Step 4 – Automation & Apps

- Create any space specific automation rules such as copying data from parent to child.
- Connect relevant integrations such as Confluence, Figma, or Git.
- Configure applicable 3rd party marketplace apps such as Xray.

Step 5 – Access

- Add users to your space as space administrators (can make space settings changes)
- Add users to your space as users (if you selected limited or private).

Team-Managed Space Checklist

1: Space Basics

- [ ] Define the space name and key (can be changed later, but breaks filters).
- [ ] Select the space template (Scrum, Kanban, Bug Tracking, or Business/Work Management Type).
- [ ] Decide if this space should be team-managed because you don't need to scale or share configs

2. Roles & Permissions

- [ ] Assign a space lead (responsible for ownership).
- [ ] Set up space roles (Administrators, Members, Viewers).
- [ ] Determine Access Level (Open, Limited, Private)

3. Work Item Types & Workflows

- [ ] Decide which work item types to use (Epic, Story, Task, Bug, Sub-task, custom).
- [ ] Customize workflows (statuses + transitions) to reflect how your team works.
- [ ] Keep it simple — avoid too many statuses that confuse reporting.
- [ ] Add custom fields (if needed) but don't overload.

4. Boards & Backlog

- [ ] Add additional columns (statuses are automatically created)
- [ ] Define sprints (for Scrum teams).
- [ ] Define columns on the board (map statuses logically).
- [ ] Add swimlanes or quick filters if helpful.

5. Automation & Rules

- [ ] Add basic automations (e.g., auto-assign work items, transition on resolution).
- [ ] Decide on notifications (who gets email updates, when).
- [ ] Consider rules for sub-tasks, linked work items, or dependencies.

6. Reports & Metrics

- [ ] Enable reports (burndown, velocity, cumulative flow, cycle time).
- [ ] Decide what metrics matter most and build filters and dashboards
- [ ] Set expectations on how reporting will be used (e.g., retrospectives, management updates).

7. Integrations & Apps

- [ ] Connect with Confluence (for documentation).
- [ ] Integrate with Bitbucket/GitHub if it's a dev team.
- [ ] Enable Slack/Teams/Jira mobile app for quick updates.

8. Space Settings & Governance

- [ ] Decide on naming conventions (for work items, versions, components).
- [ ] Clarify who can create new work types/fields (avoid chaos).
- [ ] Confirm data retention & archiving practices (how long will work items live).
- [ ] Align on team agreements (when to move work items, when to close).

Setting Up a Company-Managed Space Guide

Step 1 – Create the space

- On left hand side vertical menu go to spaces and click + (Create space). ONLY Jira Administrators and above will be able to create company-managed spaces.
- Select the space template you need (Software Development for Agile project management or Software development).
- Select Kanban or Scrum (Scrum if you need to have a Sprint).
- Click on Use Template (This just shows you the features of a Scrum space template).
- Click on Select a company-managed space (The BLUE button).
- Enter space name and key.
- Determine if you want to share settings with an existing space.
  - This allows you to use a "custom template" if you don't have Enterprise.
  - You need to have permissions to access the space you'll use to share the settings.
  - This new space and the space you select to share the settings will now share settings which means any changes to the configurations now change for both spaces.
- Click on Next.
- Skip the step where you invite users because this may add licenses ($$) to your Jira.
- For now, skip the Code and space Pages section. Just click Continue at the bottom.

Step 2 – Configure Space Settings (You need be a Jira Admin to make any of these changes)

- Click on the three dots (...) next to space name → space Settings.
- Under Work Items → Types, Add/Edit/Remove Work Types.
- Under Work Items → Screens, Add/Edit/Remove any custom fields.
- Under Workflows, Add/Edit/Customize Workflows.
- Optionally, add any work type level security (Who can create/edit specific work items).
- Optionally, add any versions or components that will be used by your teams.

Step 3 – Automation & Apps

- Create any space specific automation rules such as copying data from parent to child.
- Connect relevant integrations such as Confluence, Figma, or Git.
- Configure applicable 3rd party marketplace apps such as Xray.

Step 4 – Permissions

- Add users to your space as space administrators (can make space settings changes).
- Add users to your space as users (based on your Browse space permissions).

Step 5 – Board Setup

- Click on the three dots (...) next to your board name → Board Settings.
- Configure board columns and statuses (Column name doesn't have to match Status name).
- Add any custom filters if needed.
- Configure a default swimlane for the board.
  - Users can select a group by at the board level that allows them to dynamically pick the swimlane view but it's more limited.
- Configure the timeline view (schedule work by sprint or independently).
- Configure estimation details (pick between using Story Points or Original Estimate (hours)).

Company-Managed Space Checklist

1: Space Basics

- [ ] Define the space name and key (can be changed later, but breaks filters).
- [ ] Select the space template (Scrum, Kanban, Bug Tracking, or Business/Work Management Type).
- [ ] Decide if space should be company-managed because you do need to scale or share configs.

2. Roles & Permissions

- [ ] Assign a space lead (responsible for ownership).
- [ ] Set up global space roles (Administrators, Developers, Users, Customers/Vendors).
- [ ] Customize Permission Scheme to control who can view the space, edit/create work items, etc).

3. Work Item Types & Workflows

- [ ] Decide which work types to use (Epic, Story, Task, Bug, Sub-task, custom).
- [ ] Customize workflows (statuses + transitions) to reflect how your team works.
- [ ] Keep it simple — avoid too many statuses that confuse reporting.
- [ ] Add custom fields (if needed) but don't overload.

4. Boards & Backlog

- [ ] Enable backlog view (for Kanban if the team needs to prioritize work).
- [ ] Define sprints (for Scrum teams).
- [ ] Define columns on the board (map statuses logically).
- [ ] Add swimlanes or quick filters if helpful.

5. Automation & Rules

- [ ] Add basic automations (e.g., auto-assign work items, transition on resolution).
- [ ] Decide on notifications (who gets email updates, when).
- [ ] Consider rules for sub-tasks, linked work items, or dependencies.

6. Work Item Types & Schemes

- [ ] Select or create an Work Type Scheme (Epic, Story, Bug, Task, Sub-task, Feature, etc.).
- [ ] Decide if you need custom work types (e.g., Risk, Change Request).
- [ ] Apply an Work Type Field Configuration Scheme (which fields are required, hidden, or optional).

7. Integrations

- [ ] Connect with Confluence (for documentation).
- [ ] Integrate with Bitbucket/GitHub if it's a dev team.
- [ ] Enable Slack/Teams/Jira mobile app for quick updates.

8. Space Settings & Governance

- [ ] Decide on naming conventions (for work items, versions, components).
- [ ] Clarify who can create new work types/fields (avoid chaos).
- [ ] Confirm data retention & archiving practices (how long will work items live).
- [ ] Align on team agreements (when to move work items, when to close).

Setting Up Jira for SAFe Teams

1. Define the Hierarchy (Work Item Types)
In SAFe, work flows from strategy to execution. Jira's work types should reflect this:

Portfolio / Strategy Level
- Theme / Initiative (Often tracked in Jira Align or Advanced Roadmaps/Plans)
- Epic → Large cross-cutting work, broken into Features. (A new work type named Epic)

Program (ART) Level
- Feature → Value delivery that fits within a PI (Program Increment). **(You need to rename your epic to Feature)**
- Enabler → Technical work needed to support features.

Team Level
- Story → End-user or technical story.
- Task / Subtask → Execution-level work items.
- Bug → Defect handling.

In Jira, you can configure this hierarchy and leverage in Advanced Roadmaps:
- Initiative → Epic (New Custom Work Type) → Feature (Renamed Epic) → Story → Subtask.

2. Workflows & Statuses
You'll want different workflows per level (Portfolio, Program, Team), but they should stay consistent enough to enable reporting.

Epic Workflow (Portfolio level)
Statuses:
- Funnel → Reviewing → Implementing → Done

Feature Workflow (Program level)
Statuses:
- To Do → Ready for PI Planning → In PI → In Progress → Done

Story / Task Workflow (Team level)
Statuses:
- To Do → In Progress → In Review → Done

Bug Workflow (Team level)
Statuses:
- To Do → In Progress → In Testing → Done

Best practice: Keep workflows simple. Avoid too many statuses, but make sure there's a clear handoff between PI planning, execution, and completion.

3. Fields & Configurations

- Fix Version → Used to represent Program Increments (PIs).
- Components → Can represent ARTs, value streams, or subsystems.
- Labels → Quick tagging for cross-cutting concerns.
- Custom Fields (recommended):
  - WSJF (Weighted Shortest Job First) → for prioritizing Features/Epics.
  - PI Objective → summary of planned value delivery.
  - Commitment Flag → to record whether a Feature/Story was committed vs. uncommitted in PI planning.

4. Boards & Backlogs

- Team Boards → Scrum/Kanban boards for execution at story/task level.
- Program Board (Jira Plans) → Shows Features across teams, dependencies, and PI commitments.
- Portfolio View → Epics and Initiatives in Plans

5. Program Increment (PI) Configuration

- Create Fix Versions named like PI-2025.1, PI-2025.2.
- During PI planning, assign Features and Stories to those versions.
- Use Plans to plan capacity across multiple teams.

6. Reporting & Metrics

- Portfolio Level → Progress of Epics across PIs.
- Program Level → Feature burndown, Program Board, dependency tracking.
- Team Level → Velocity, Sprint burndown, cumulative flow.
- Custom Dashboards → Business value delivered, predictability (commitment vs. done).

7. Governance / Best Practices

- Use Company-Managed spaces (not Team-Managed) for SAFe — you need consistent workflows.
- Standardize work type hierarchy across all teams.
- Train teams on linking Epics → Features → Stories correctly.
- Decide whether to use Jira Align (if your org is fully SAFe) or Plans (for mid-scale SAFe).
- Limit customization per team — consistency is critical for roll-up reporting.

Pro Tip: If you don't have Jira Align, you can still approximate SAFe in Jira Software + Plans with:

- Epics (Portfolio) → Features (Program) → Stories (Team) → Sub-tasks.
- Use Versions = PIs, Boards = Teams, Roadmaps = ART planning.

Jira Space Glossary

Core Jira Concepts

- **Jira** – Atlassian's space and work tracking platform.
- **Space** – A collection of work in Jira, usually representing a product, service, or team.
- **Work Item** – The fundamental piece of work in Jira (task, bug, story, etc.).
- **Epic** – A large body of work broken down into stories or tasks.
- **Story** – A user-focused requirement or feature request.
- **Task** – Some piece of work someone on the team needs to complete.
- **Bug** – A work type representing a defect.
- **Sub-task** – Smaller work item under a story/task/Bug.
- **Feature** – A higher-level work item (often custom work type in SAFe setups).
- **Initiative** – Strategic high-level objective, above epics.

Fields & Properties

- **Summary** – Short title/description of a work item.
- **Description** – Detailed explanation of a work item.
- **Assignee** – The person responsible for completing the work item.
- **Reporter** – The person who created the work item.
- **Priority** – Importance of a work item (Highest → Lowest).
- **Resolution** – Final outcome of a work item (Done, Won't Fix, Duplicate, etc.).
- **Status** – Current state of a work item (To Do, In Progress, Done).
- **Labels** – Free-text tags to categorize work items.
- **Components** – Subsections of a space or system (modules, teams, services).
- **Fix Version** – The release/version a work item is planned for.
- **Affects Version** – Version where a bug/issue was found.
- **Due Date** – Deadline for completing the work item.
- **Environment** – Technical context (OS, browser, system).
- **Parent** – Connects a child to higher-level work item. (Subtask → Story, Story → Epic, Epic → Initiative)

Workflows & Transitions

- **Workflow** – Set of statuses and transitions a work item goes through.
- **Transition** – Movement of a work item between statuses.
- **To Do** – Default starting status.
- **In Progress** – Active work status.
- **Done** – Completed work status.
- **Workflow Scheme** – Mapping of work types to workflows.
- **Resolution Screen** – Screen displayed when resolving a work item.
- **Condition** – Restriction on a workflow transition.
- **Validator** – Rules that must be met before a transition.
- **Post Function** – Automation triggered by a workflow transition.

Roles & Permissions

- S**pace Lead** – Default owner of a space.
- S**pace Role** – Group of users (Admins, Developers, Viewers).
- **Permission Scheme** – Defines what actions users can take.
- **Notification Scheme** – Defines who gets emails and when.
- **Security Scheme** – Controls visibility of work items.
- **Browse space** – Permission to view a space's work items.
- **Assign work** – Permission to assign work.
- **Transition work** – Permission to change a work item's status.
- **Administrators** – Users who manage configurations.
- **Developers** – Users who work on work items.
- **Viewers** – Read-only access role.

Boards & Agile

- **Board** – Visual representation of work items.
- **Scrum Board** – Used for sprint-based work.
- **Kanban Board** – Used for continuous flow of work.
- **Backlog** – List of work not yet in progress.
- **Sprint** – Time-boxed period for delivering work.
- **Velocity** – Measure of completed story points per sprint.
- **Burndown Chart** – Graph showing remaining work in a sprint.
- **Cumulative Flow Diagram** – Visualization of work across statuses.
- **Epic Burndown** – Progress of an epic over time.
- **Swimlane** – Horizontal grouping of work items on a board.
- **Quick Filter** – Custom filter on boards.
- **WIP Limit** – Maximum allowed work in progress.
- **Estimation** – Method of sizing work (Story Points, Time).
- **Capacity** – Planned workload for a sprint/team.

Searching & Filters

- **Filter** – Saved search query.
- **JQL (Jira Query Language)** – Advanced query language for search work items based on field data.
- **Search** – Finding work items in Jira (basic or advanced).
- **Saved Filter** – Reusable search criteria.
- **Filter Subscription** – Automatic email report of filter results.

Reports & Dashboards

- **Dashboard** – Customizable page with gadgets.
- **Gadget** – Widget showing reports, charts, or filters.
- **Pie Chart** – Work Items grouped by a field (status, assignee).
- **Two-Dimensional Gadget** – Matrix of work items by two fields.
- **Heatmap** – Visualization of workload distribution.
- **Created vs. Resolved Chart** – Trend of new vs. completed work items.
- **Control Chart** – Cycle time visualization.
- **Velocity Chart** – Team's delivery rate across sprints.

Configurations

- **Work Type Scheme** – Defines which work types are available in a space.
- **Field Configuration** – Controls which fields are visible/required.
- **Screen Scheme** – Defines which screens are used for work item operations.
- **Screen** – Collection of fields shown to users (Create/Edit/View).
- **Custom Field** – User-defined data field.
- **Field Context** – Scope of a custom field.
- **Resolution Scheme** – Set of possible resolutions.
- **Notification Event** – Trigger for sending notifications.

Miscellaneous

- S**pace Category** – Grouping of spaces.
- **Archived space** – space no longer active.
- **Cross-space Board** – Board spanning multiple spaces.
- **Epic Panel** – Backlog view of epics.
- **Dependency** – Relationship between work items.
- **Link** – Connection between work items (blocks, relates to).
- **Watcher** – User subscribed to updates on a work item.
- **Time Tracking** – Estimated vs. logged work.
- **Work Log** – Record of time spent on a work item.
- **Reindexing** – Jira admin task to refresh the search index.

Setting Up a Jira Space for Software Teams Runbook

1. Work Item Types (Formerly work Types)

Keep it lean but aligned with Scrum:

- Epic → Large body of work, spans multiple sprints.
- Story → A user story (end-user functionality).
- Bug → A defect that needs fixing.
- Task → Technical or non-functional work.
- Sub-task → Small pieces of a story/task.

2. Workflows & Statuses

Epic/Subtask Workflow

To Do → In Progress → Done

Story/Task Workflow

To Do → In Progress → Code Review → Internal Testing → Done

Bug Workflow

New → To Do → In Progress → Code Review → Internal Testing → Done

3. Key Fields

Story Points → Estimation field for Scrum teams.
Assignee → Who's working on it.
Sprint → To tie work to a timebox.
Fix Version → For release planning.
Components (optional) → Use if you want to tag by module or subsystem.
Priority → Helps teams order work (but usually backlog order is the main driver).
Pro Tip: You may want to consider having different fields per work item type, just like with your workflows

4. Boards & Backlog

- Scrum Board for each team.
- Add swimlanes by Epic or Story if useful.
- Enable Quick Filters (e.g., "My work," "Bugs Only," "Unestimated").

5. Sprints & Cadence

- Create sprints directly in the backlog.
- Define sprint length (common = 2 weeks).
- Use fix versions for releases that may span multiple sprints.
- Add Definition of Ready (DoR) and Definition of Done (DoD) as team agreements (document in Confluence or space sidebar).
- Add Acceptance Criteria so developers know how to test their code and QA knows what to test against

6. Automation Ideas

Auto-assign Bugs to component lead.
Send Slack/Teams notification when a sprint starts/ends.

7. Reports & Metrics

For Scrum teams, enable:

- Burndown Chart (per sprint).
- Velocity Chart (for forecasting).
- Sprint Report (for completed vs. carried-over work).
- Epic Burndown (progress on larger initiatives).
- Cumulative Flow Diagram (to check WIP health).

Setting Up a Jira Space for Marketing Teams Runbook

1. Work Item Types (Formerly Issue Types)
Marketing teams need work types that reflect their workflow:
Epic → Large campaign (e.g., "Holiday 2025 Campaign").
Story → Marketing deliverable (e.g., "Launch email sequence," "Landing page design").
Task → Supporting work (e.g., "Write blog post copy," "Design banner").
Bug/Defect → QA works (typos, broken links, formatting errors).
Sub-task → Tiny to-dos within a task (e.g., "Add CTA link," "Proofread copy").

2. Workflows & Statuses
Marketing workflows often need review and approval steps:
Epic Workflow (Campaigns) / Sub-task
To Do→ In Progress → Done
Story/Task Workflow (Deliverables)
To Do → In Progress → In Review → Approved → Done
Bug Workflow (QA) / Defect
New → To Do → In Progress → Ready for Review → In Review → Done
Pro Tip: Include In Review / Approved steps since marketing involves stakeholders (legal, brand, management).

3. Key Fields
Due Date → Critical for campaign deadlines.
Assignee → Who's responsible.
Priority → High/Medium/Low urgency.
Labels → Campaign tags (e.g., "SEO," "Paid Ads," "Social").
Components → Channel or team (Email, Social Media, Events, Content, Design).
Fix Version → Optional; can represent campaign launch dates.
Custom Fields (recommended):

- **Campaign Objective** (short description of goal).
- **Content Type** (blog, email, ad, video, etc.).
- **Approval Owner** (who signs off).

4. Boards & Backlog
Kanban Board works best for continuous campaign work.
Columns should match workflow (To Do → In Progress → In Review → Approved → Done).
Swimlanes by Campaign (Epic) or by Channel (Component).
Quick filters: "Needs Review," "Overdue," "My Work."

5. Automation Ideas
Auto-notify approver when task moves to "In Review."
Send Slack/Teams alert for tasks due in 3 days.
Auto-label new work items based on Component (e.g., Social = auto "Social Media" label).

6. Reports & Metrics
Marketing teams care about throughput and deadlines more than velocity:
Cumulative Flow Diagram → Tracks work in each stage.
Control Chart → Cycle time for content creation.
Created vs. Resolved Chart → Workload trend.
Calendar View → Visual of campaign deadlines.
Epic Report → Campaign-level progress.

Setting Up a Jira Space for Finance Teams Runbook

1: Work Item Types (Formerly Issue Types)
Initiative / Project → Major finance efforts (budgeting cycle, annual audit, system implementation).
Task → General finance activities (month-end close, reconciliation, vendor payment).
Approval → Used for expense approvals, purchase requests, or investment decisions.
Bug/Defect (optional) → For finance system errors or compliance gaps.
Request → When other departments submit finance-related requests (new vendor setup, invoice processing, etc.)
Sub-task → Smaller steps within a task (upload report, verify entry, review approval).

2: Workflows
Finance work usually has a mix of process-heavy tasks and request-driven work. A solid workflow might look like:
General Finance Task Workflow:
To Do → In Progress → In Review → Approved → Done
Request/Approval Workflow:
To Do → In Review → Pending Approval → Approved → Rejected
Initiative / Bug Workflow:
To Do → Investigating → In Review → Resolved → Done
Sub-task Workflow:
To Do → In Progress → Done

3: Custom Fields
Cost Center / Department → who owns the expense/project
Budget Code → align with finance reporting codes
Amount / Value → financial impact of the item
Due Date → especially critical for financial close
Approval Required? (checkbox or dropdown)
Vendor Name → when dealing with procurement

4: Permissions
- Strong restrictions (e.g., only finance managers can move work items to Approved).
- Limited edit access once items are approved (audit trail).

5: Automation
- Auto-assign requests based on cost center or type.
- Send Slack/Email reminders for Pending Approval.
- Transition to Approved when a manager approves via Jira comment/button.

6: Boards & Views
- Kanban Board → for ongoing finance work.
- Calendar View → for due dates (month-end, quarterly reporting).
- List View → for quick task management.

7: Reports & Dashboards
- Tasks by Cost Center / Department
- Pending Approvals
- Budget Requests vs. Approved
- Monthly Close Progress

Setting Up a Jira Space for Sales Teams Runbook

1: Work Item Types (Formerly Issue Types)
Opportunity / Deal → Core sales object, representing a potential sale.
Lead → New inbound/outbound prospect to qualify.
Task → General activities (send contract, schedule demo, update CRM).
Sub-task → Specific steps under a deal/task (draft proposal, collect signatures).
Renewal → For customer contract renewals.
Request → Internal requests to Sales Ops (pricing exception, discount approval, legal review).

2: Workflows
Sales workflows mimic a CRM pipeline but can be extended for approvals & handoffs.
Opportunity/Deal Workflow:
Lead Identified → Qualified → Proposal Sent → Negotiation → Pending Approval → Closed Won / Closed Lost
Internal Sales Request Workflow (pricing/discount/legal):
Submitted → In Review → Approved → Rejected
Renewal Workflow:
Upcoming Renewal → Customer Contacted → Negotiation → Closed Won / Closed Lost

3: Custom Fields
Account / Company → link to customer
Deal Value → $ amount
Close Date → expected close
Stage → mirrors pipeline (Lead, Qualified, etc.)
Sales Rep → owner of the deal
Probability % → confidence of winning
Competitor → (optional) track competition
Approval Required? → for discounts or special terms

4: Permissions
- Only Sales Managers can move deals to Closed Won.
- Reps can edit deals but not delete them (audit trail).

5: Automation
- Auto-assign leads based on territory.
- Trigger Slack/Email reminders if a deal sits in a stage too long.
- Notify Finance when a deal is Closed Won.
- Auto-create Renewal Tasks 90 days before contract expiration.

6: Boards & Views
Kanban Board → visualize deals across stages.
Calendar View → track close dates & renewal dates.
List View → manage high-volume leads.
Dashboard View → show sales pipeline metrics.

7: Reports & Dashboards
- Pipeline by Stage (deal count + value).
- Deals Closing This Month/Quarter.
- Win Rate by Rep.
- Average Sales Cycle Length.
- Renewals Due Soon.

Jira Spaces Creating Mistakes To Avoid

Setting up your Jira spaces incorrectly can cost you greatly in both time and efficiency. Avoid the following mistakes and learn from all the mistakes that myself and many other Jira admins have already done.

1. Choosing the Wrong Space Template

- Mistake: Picking "Team-managed" when the team really needs "Company-managed" (or vice versa). It's easy to fall into the Team-managed trap because Atlassian wants you use this space type. In fact, when you are launching Jira for the very first time, you don't really have an option. Remember, if you want the full Jira experience and scalability, you want to select a company-managed space.
- **Pro-tip**: Decide based on governance needs. Team-managed = lightweight, decentralized. Company-managed = standardization, reporting across teams.

2. Overcomplicating Workflows

- Mistake: Adding 15+ statuses (e.g., "In Progress," "Working On It," "Ongoing," "Doing Work"). Too many teams get caught up with "Ready for X" statuses. You have to find the right balance. Not every step in your process needs a dedicated status in your Jira workflow.
- **Pro-tip**: Keep it simple. Start with To Do → In Progress → Done and only add stages when there's a real business rule tied to it.

3. Too Many Custom Fields

- Mistake: Creating a new field for every idea ("Customer Industry," "Customer Vertical," "Customer Segment"). Suddenly, you've got 200 fields nobody uses. You especially want to be careful with duplicate fields and when in doubt, use a field context to avoid having to create an unnecessary duplicate field.
- **Pro-tip**: Use only the fields that actually drive reporting, automation, or approvals. Less is more.

4. No Clear Work Type Hierarchy

- Mistake: Everything is a "Task." Teams don't distinguish between Epics, Stories, Bugs, or Sub-tasks. Reporting becomes chaos. Jira is three-dimensional. It has a 3 tier hierarchy out of the box and for premium/enterprise subscribers, you can add more levels to help organize your work.
- **Pro-tip**: Not all work is created equal. Leverage your hierarchies to bucketize your work and help break down requirements. Often times stories or tasks are too big in scope and lead to teams rolling work over because they tried to define too many requirements at the story/task level.

5. Forgetting About Permissions

- Mistake: Giving everyone full admin rights. Users can accidentally delete workflows, boards, or fields.
- **Pro-tip**: Use global space roles to define a single space scheme but can be tailored by space

Common Jira Work Types Guide

These are available in nearly every Jira space. Think of them as the building blocks.

Epic
Big body of work (weeks/months).
Example: “Launch new website”

Story
User-focused requirement.
Example: “As a user, I want a search bar so I can find products quickly.”

Task
General unit of work not tied to a user story.
Example: “Set up new dev environment.”

Sub-task
Breaks down a Story/Task into smaller chunks.
Example: “Write test cases for login API.”

Bug
Defect or error in the system.
Example: “Checkout button not responding on mobile.”

These are custom work item types that you should consider adding to your Jira

Spike
Research or investigation task.
Example: “Research GraphQL API performance limitations.”

Improvement
Enhancement to an existing feature.
Example: “Optimize query response time on product search.”

Change Request
Request to modify scope, functionality, or process.
Example: “Switch hosting provider for staging environment.”

Initiative
A large body of work that spans multiple epics, usually representing a business outcome.
Example: “Launch Subscription Billing Platform” or “Implement Enterprise Data Lake.”

Theme
A broad, strategic focus area that aligns with business goals. Think of it as a "bucket" of work that ties back to company strategy.
Example: “Digital Transformation” or “Cloud Migration.”

Feature
A service or capability that delivers value to the customer and can be delivered by one Program Increment (PI) or release.
Example: “Enable Credit Card Payments” or “Add Single Sign-On.”

Jira Workflows Best Practices Guide

Workflows is where I see teams mess up the most. It's easy to get overwhelmed with creating the perfect workflow that you often make critical mistakes that end up hurting the team in the long term and the workflow doesn't scale as your team evolves. Let's discuss some common best practices that you should be considering whenever making a workflow.

1. Keep It Simple

- Avoid overengineering—start with the fewest statuses and transitions possible.
- Train your team to use the global transitions as opposed to over complicating your workflow to force your team into a specific transition path.
- Add complexity only when the team has a clear need.
- A typical agile team rarely needs more than To Do → In Progress → Done plus 1–2 other key statuses.

2. Use Clear & Action-Oriented Status Names

- Statuses should describe the state of work, not people's roles.
- Examples:
  - Good: In Review, Blocked, Ready for Testing
  - Bad: With QA, Dev Done, Waiting on Bob

3. Avoid Duplicate or Redundant Statuses

- Don't create Testing, QA Testing, and Verification unless each has a distinct purpose.
- Too many similar statuses confuse reporting and transitions.
- Avoid the "Ready For" statuses unless your team really needs the extra status.

4. Design for Reporting

- Think about how workflow design affects Jira reports, dashboards, and burndown charts.
- Example: If you add a Ready for Release status, make sure it still counts toward Done when reporting sprint velocity.

5. Use Conditions, Validators, and Post-Functions Wisely

- Conditions: Restrict who can transition work items (e.g., only QA can move to Tested).
- Validators: Require fields before moving forward (e.g., must add "Fix Version" before Ready for Release).
- Post-functions: Automate repetitive tasks (e.g., auto-assign to QA when moving to In Testing).

6. Map Workflows to Real Processes

- Align workflows with how teams actually work—not an idealized process.
- Interview the team to see where items get stuck or handed off.

7. Use Workflow Schemes

- Don't give every space a unique workflow unless necessary.
- Standardize workflows across spaces where possible (e.g., all software teams share one workflow).

Jira Workflows Mistakes to Avoid

Mistakes will always happen, but it's better to learn from the mistakes that others have done and try to avoid making the mistake yourself. In other words, learn from my mistakes and you can be a better Jira admin because you don't have to repeat.

The Most Critical Mistake Jira admins make is renaming a status in Jira. This is a global change in Company-managed spaces and should be avoided at ALL COSTS!

1. Overcomplicating the Workflow

- Dozens of statuses like In Review by QA, QA Verified, Waiting for Release, Ready for Deployment.
- Every team having its own version of the same workflow.
- Every work item type having its own version of the same or similar workflow
- Different workflows that are very similar having similar statuses for essentially the same thing
  - Example: In Progress, In Development are too close that basically describe that work is in flight by someone.
- Why it's bad: Creates confusion, makes reports unreadable, and slows teams down.

2. Not Using the Resolution Field Properly

- Leaving work in "Done" without setting a Resolution.
- Having a Resolution like Fixed, Duplicate, Will Not Fix but forgetting to apply it.
- Setting the Resolution with a post function, but then prompting your user to also set the Resolution which is always over written by the post function
- Not clearing the Resolution when you transition out of your "Done" status
- Why it's bad: Jira reports (burndown, velocity, done vs not done) break if Resolutions aren't consistent.

3. Allowing Too Many Transition Paths

- Connecting every status to every other status ("spaghetti workflows").
- Using a bad combination of global transitions and explicit transitions
- Why it's bad: Users get overwhelmed, make mistakes, and move work randomly.

4. Not Testing Before Rollout

- Making changes directly in production spaces.
- Deploying without team input.
- Not using a sandbox to test out workflow status names, transition names, or advanced post functions/automations.
- Why it's bad: You risk breaking active sprints, automations, or confusing everyone overnight.

5. Using Workflows as a Micromanagement Tool

- Adding extra statuses to monitor every handoff ("Dev Started", "Dev Halfway", "Dev Almost Done").
- Why it's bad: Creates friction, wastes time, and reduces trust in teams.

Jira Advanced Workflows Guide

1. Understand the Purpose of a Workflow
A workflow is not just a flowchart — it's a contract of how work progresses. Advanced workflows should balance structure vs. flexibility:

- Too rigid → slows teams down.
- Too loose → no accountability or visibility.

2. Advanced Workflow Patterns
Use Conditions and Transition screens to allow work to split (e.g., Dev + QA in parallel).

- Example: "In Progress" can move to Code Review or Testing based on team roles.

Add statuses like Awaiting Approval and use Conditions so only managers or approvers can transition forward.

- Example: Code ready to deploy can only be deployed by Software Dev Lead

Add Post Function to auto-notify approvers.
Add Validators to ensure field data is filled out by user
Add Conditions to control transition paths based on specific use cases

- Example: Work can transition to Escalated when Priority = Highest

Automate release transitions with CI/CD triggers.

3. Automation in Workflows
Use Post Functions to:

- Auto-assign to reporter's manager.
- Auto-set resolution.
- Update sprint or fix version.

Use Automation Rules (instead of over-engineering workflows) for:

- Auto-closing stale work items.
- Reopening bugs if linked ticket is reopened.
- Sending Slack/email updates.

4. Use Screens to Capture Information
In combination with the post functions, use specialized screens to capture specific field data as your team transitions from status to status. Not all information is known at the beginning, so it makes sense to ask for a Due Date once work moves to "In Progress"

5. Example Advanced Workflow
Backlog → Selected for Development → In Progress → Code Review → Testing → Ready for Release → Done
Validators: Ensure Acceptance Criteria is filled before "In Progress."
Conditions: Only QA can move tickets into "Testing."
Post Functions: Set resolution when transitioning to "Done."
Automation: Auto-transition from "Ready for Release" → "Done" when build pipeline succeeds.

Jira Status Tips

Every team is going to be different. They have their own way of doing work and I'm not going to write this and tell you that there is some magical pill you can take and you'll have the perfect workflow. Instead, I'm going to show you something that I do for the teams that I work with. Whatever their statuses are, it's important to write them down and define what each status (and transition) means. Taxonomy is super important and many teams become frustrated with Jira not because it's overly complicated, but because they are overwhelmed with not knowing what all the different Jira elements mean. Create a Confluence page and define each status so that every person in your company, no matter the role, has the same basic understanding of what each status means.

Status Name	Definition
To Do	Work is prioritized and ready for the team to start.
Backlog	Work has been identified but is not yet prioritized for action.
In Progress	Work is actively being done.
In Review	Work is completed but is product owner/manager review.
Code Review	Pull Request has been created
Testing / QA	The work is being tested to ensure it meets requirements.
Blocked	Work cannot move forward due to a dependency, problem, or external factor.
Done	Work is completed, and the work item has a resolution set.

Jira Fields Tips

Every team is going to be different and will need different data to drive their business. Jira ships with default fields that are very helpful, but more likely than not, you'll need to create and define your own set of custom fields. While every team is going to have different needs and there is no set list of custom fields you should use, I'm going to show you something that I do for the teams that I work with. Whatever their fields are, it's important to write them down and define what each field means. Taxonomy is super important and many teams become frustrated with Jira not because it's overly complicated, but because they are overwhelmed with not knowing what all the different Jira fields mean. Create a Confluence page and define each field so that every person in your company, no matter the role, has the same basic understanding of what each field means. I'm going to skip the default, out of the box fields, but you'll want to make sure every field in your process is defined.

Pro-tip: Break up your field definition by work item type

Field Name	Definition
Business Value	Numerical score to prioritize work.
Risk Level	Dropdown: Low / Medium / High.
Target Release / Delivery Date	Helps with planning.
Stakeholder	Person or team responsible.
Effort / T-Shirt Size	S, M, L, XL, or numeric estimation.
Cost / Budge	Estimated spend for the task.
RAG Status	Red, Amber, Green health check.
QA Owner	Person responsible for testing.
Acceptance Criteria	Requirements for the work item

Story Points Guide

T-Shirt Size	Fib #	Definition
XS	1	Something super easy to complete, very trivial, and requirements are known, no dependencies, no ambiguity, straight forward
S	2	Easy to complete, trivial, most, if not all requirements known, no dependencies, assignee might have a question or two about what it takes to complete this item.
M	3	Fairly easy to complete, somewhat trivial, some requirements are not known. Some intra-team dependencies, assignee will need to do some light follow up to get clear picture of what is being asked to complete this item
L	5	Can be completed within a sprint. Few requirements are known, more investigation will need to be done. Some internal/external dependencies. Work is somewhat ambiguous and follow ups will be needed
XL	8	Will take most, if not all of the sprint to complete. Little, to no requirements defined. Will required dependencies internally and externally. This item will most likely be at risk of not being completed within a single sprint if team isn't proactive toward addressing unknowns.
XXL	13	Not achievable in a single sprint. Something super complex to complete, not trivial at all, no requirements are known, multiple dependencies, very ambiguous, requires a lot of discussion and strategy to figure out. This needs to be broken down significantly.

Jira Custom Fields Mistakes to Avoid

1. Creating Duplicate Fields

- **Example**: Multiple teams each create their own “Due Date,” “Target Date,” or “Delivery Date” fields.
- **Why it’s bad:** Causes reporting chaos—users don’t know which field to use, dashboards show inconsistent data, and maintenance is harder.
- **Fix:** Always search before creating a new field. Use field contexts instead of duplicates.

2. Too Many Custom Fields

- **Example:** A Jira instance with 2,000+ custom fields (I’ve seen it).
- **Why it’s bad:** Performance suffers (searches, screens, indexing). Users get overwhelmed with irrelevant fields.
- **Fix:** Limit fields to business-critical data. Audit and delete unused ones regularly.

3. No Naming Conventions

- **Example:** “Due Date,” “duedate,” “Target Delivery,” and “ETA” all exist.
- **Why it’s bad:** Confuses users, creates inconsistencies in JQL, dashboards, and automation.
- **Fix:** Use prefixes or suffixes (e.g., Finance Invoice Number, Marketing Campaign Name).

4. Not Using Field Contexts

- **Example:** Creating one global dropdown with 200 values for all spaces.
- **Why it’s bad:** Most users see irrelevant options, cluttering the UI. Performance is hit.
- **Fix:** Restrict custom fields to specific spaces/work types using contexts.

5. Creating Fields Instead of Using Versions/Components/Labels

- **Example:** Making a custom field called “Region” when Components or Labels would do the job.
- **Why it’s bad:** Increases field bloat and complexity unnecessarily.
- **Fix:** Always ask: “Can an existing Jira feature (components, labels, versions) solve this?”

6. Using Free-Text Fields Too Often

- **Example:** “Customer Name” as a text field. Users enter “IBM,” “I.B.M.,” and “ibm.”
- **Why it’s bad:** Leads to messy, unreportable data.
- **Fix:** Use dropdowns, user pickers, or cascading selects for consistency.

7. No Ownership or Governance

- **Example:** Fields are created by request, but no one knows who owns them or why they exist.
- **Why it’s bad:** Orphaned fields pile up and never get cleaned.
- **Fix:** Maintain a field registry (in Confluence) with owner, purpose, and usage.

8. Not Considering Reporting Needs

- **Example:** A text field “Risk Level” where people type in “High,” “HIGH,” “H.”
- **Why it’s bad:** Impossible to filter or build meaningful dashboards.
- **Fix:** Use controlled field types (select lists, radio buttons, numbers) where possible.

9. Custom Fields for Temporary Use

- **Example:** A project creates “Event 2024 ID” and then never uses it again.
- **Why it’s bad:** Adds clutter to the instance forever.
- **Fix:** If temporary, document and schedule deletion/archiving.

How to Pick the Right Custom Fields Type?

When creating a new custom field, there are many different types available. Picking the right type is both a little science and a little art. This guide is designed to help you determine which custom field type you should select. It will show you questions you should ask the individual requesting a custom field so that you pick the right type for the job.

Gate Checks — Do you even need a new field?

- Reuse first: Does a standard field (Priority, Fix Version, Components, Labels, Assignee, Due date, Environment) already cover it?
- Existing custom field: Does one already exist with the same meaning? (Search admin → fields.)
- Structure over text: Will this drive reporting/automation? If yes, prefer structured types (Select/Drop Down) over free text.
- Context, not global: If you still need it, plan field context (which spaces/work types) before creating.

Define the single question the field must answer

- Write one sentence: "This field tells us _____ so that we can _____."

If you can't finish that sentence, don't create the field.

Choose the data shape (drives field type)

- **Binary (yes/no)?** → Select list (single choice) with Yes/No.
- **One choice from a known list?** → Select list (single choice).
- **Multiple choices allowed?** → Select list (multiple choice) (use sparingly).
- **Hierarchy (category → subcategory)?** → Cascading select (Can only do 2 levels)
- **Short free text (IDs, short codes)?** → Text field (a "tweet").
- **Long text (requirements, acceptance criteria)?** → Text field (multi-line).
- **Number, score, or currency amount?** → Number field (only allows numbers).
- **Date only (no time-of-day)**? → Date picker.
- **Date + time matters (deadlines w/ time zones)?** → Date time picker (Be careful with this one as it's not supported in Plans)
- **Person/owner/approver?** → User picker (single); multiple approvers → Multi-user picker or JSM Approvers (Users must be licensed Jira users).
- **Team/Group responsibility?** → Group picker (or Components for stable, reusable ownership).
- **Link to Jira entity?** → space picker, Version picker, or leverage Components/Labels.
- **A URL?** → URL field (Don't use this for Code links, you should configure your git repo instead and use the native git integrations).
- **Time estimates/spent?** → Prefer Jira's Time tracking (don't duplicate).
- **Flags / status toggles?** → Checkbox (or small single-select if more than Yes/No).

Decide cardinality & constraints

- Single vs multi-value? Default to single unless you truly need multi.
- Required? Only if absolutely necessary (test first—required fields can block transitions).
- Defaults? Set a sensible default for single-selects to reduce noise

Screen & Schemes Cheat Sheet

Figuring out Screens, Screen Schemes, Work Type Screen Schemes isn't for the faint of heart. On top of that, it's not always intuitive to know that when working with fields, you don't go directly to the field settings, but rather the screens. This cheat sheet is intended to help you understand how screens work and how to successfully add a field to your Jira space.

First, you want to use this cheat sheet to add a field that exists in Jira to a Jira space. If the field doesn't exist yet, then you need to create the field first.

Next, once the field exists, you have to determine which work type screen you are going to add the field to. By default, Jira software spaces have two screen schemes. One for bugs and one for everything else. If you are trying to add the field to the bug only, then your job is very easy, but if you only want to add a field to an epic, then you need to do a little bit of work first before you can add a field only to the epic.

1: Go to the gear → Work Items → Screens. From here, you can either copy an existing screen (recommended) or start with a brand new screen. I wouldn't do a new screen because then you'll have to remember to include required fields such as your Summary field. It's better to make a copy of the existing screen. Find the screen currently used by your space.

2. Copy the screen and rename it to something that logically makes sense. Before you proceed, remember that screen schemes have 3 different operations (Create, Edit, and View). If you plan on allowing your users to create, edit, and view the field, then you only need to make a single screen. Otherwise, you'll want to make the appropriate number of screens based on which operation is important to you.

3. Once the screens are all created, you need to go to Screen Schemes on the left and either copy one of your spaces existing screen scheme or create a new one. For each operations, add your screen and link to the appropriate operation.

4. Last step is to go back to your space. Under space Settings → Work items → Screens, click on the action button and select Edit screens. From there, select the work item type you want to associate with your newly created screen scheme. In this case, we want to select Epic and then select your new Epic Screen Scheme which will have the appropriate Screen which will then contain your field.

How to Make Fields Required Guide

There are a couple of different ways to make fields “required” in Jira, but there is officially only one way to add the red * to a field. This guide will show you how to make fields required without breaking Jira for everyone.

1. Go to the gear → Work Items → Field Configurations
2. Make a copy of the default field configuration
3. Rename your new copy to something that clearly tells you why you are making a new field configuration. Don’t use generic names. Make sure field configuration is in the the new name.
4. Find your new field configuration and click on Configure
5. Find the field(s) that you want to make required.
    a. Slide the Required slider to make the field required.
6. On the left side, click on field configuration schemes
7. Click on the Add field configuration scheme button.
8. Provide an appropriate name and description. Make sure you include field configuration scheme in the name.

You have two paths you can take. If you are going to make a field required for all work items, then you’ll want to follow steps 9 - 11. If you are going to make a field required for a specific work type, then skip down to step 12.

9. Click on Edit for the Default field configuration.
10. In the Field Configuration Drop down, select your newly created field configuration. Failure to make this change will result in you using the wrong field configuration.
11. Click on Update to use the new field configuration. This field configuration is going to be applied to all the work types in your space.

If you want to leave the default field configuration, but only apply your new field configuration to a specific work type, then follow the following steps.

12. Click on Associate a work type with a field configuration
13. Select your work item type and the field configuration we created previously.

Repeat for each additional field configuration.

Finally, go to the Jira space where you want to apply this new field configuration scheme.

14. Space Settings → Fields → Actions → Use a different scheme → Select your new field configuration scheme.

Now, if you try to create a new work item, you will see your required fields correctly.

Permission Scheme Breakdown

Company-managed space permissions can be overwhelming to understand. There are multiple "levers" that can be pulled to tweak the permissions you want for a space. Because permissions are in a scheme, they can be shared with other company-managed spaces. This breakdown will help you understand how Jira permission schemes work.

There are many different ways to assign permissions but here are some considerations:

- Every Jira "Software" space will use the same permission scheme out of the box. Keep this in mind so that you don't accidentally change the permissions for the incorrect space.
- Every Jira "Business" space will use the same permission scheme. Just like above, be careful with this as you'll have to swap out permissions schemes to minimize impact.
- Try to avoid making permissions schemes that are specific to just a single space. Specifically adding a user is a great way to make a permission scheme work for a limited number of spaces.
- Utilize groups, but I would recommend you only leverage the site/org/Jira admins group. For all other groups, I would recommend you use them in the people section of a Jira space and then map to a space role.
- The best practice is to use space roles in your permission scheme. This will allow you to create a one size fits many permission scheme and then each Jira space can use a combination of users and groups to leverage the permission scheme.
- Finally, don't be fooled by available space roles that aren't being used in your permission scheme. space roles are global, so every space will display them, but the most important thing for you to remember/know is that the permission scheme is what ultimately controls the permissions of your space.
- Below are the most important space permissions that need careful consideration.

Permission	Definition
Administration	This allows users to create releases, components, but also allows them to rename/rekey a space. You should minimize the number of space admins.
Browse	This is the most important permissions for end users. Without the ability to browse a space, the user cannot see the space and the work within it.
Assignable	This permission allows a user to be assigned a work item. If a user doesn't show up as a possible assignee, you want to make sure the user has this permission. If that doesn't work, there is a global permission that needs to be configured for the person doing the assignment. They need to be in a group that has the ability to search for users.
Schedule	This permission lets you set a due date for work, but more importantly, it allows you to plan work into a sprint (set the sprint field).

Permission Schemes Best Practices

- Remove any specific user unless absolutely necessary
- Leverage roles over users and groups whenever possible
- Will this permission scheme need to be used with other spaces? Heavily consider space roles.
- Grant Browse permission only if you want someone to have “read-only” access
- Grant Edit, Manage Sprint, and Schedule work permission to anyone needing to plan a sprint
- Never remove atlassian-addons-space-acces as this will break apps and other integrations
- When making a new permission scheme, make a copy of an existing one otherwise, apps will not work as you’ll be missing atlassian-addons-space-acces.
- When a user can’t assign a work item, double check the assign and assignable permission. If that still does not fix the problem, double check the global setting for searching users and groups.
- Anyone with Administer permissions will be able to change the name and key of a space which has the potential to break things.
- Some permissions need to be stacked. For example, to transition a work item, you need to be able to the the edit and transition permission.
- Follow the principle of least privilege → only give people the access they need.
- Delete work items should rarely go to anyone but admins.
- Remove permissions from inactive users, unused groups, or legacy roles. This is especially true if your permission schemes call out specific users and they are no longer with the company or team.
- Name schemes descriptively, using a generic name will cause confusion down the road
- If you have customer-facing spaces, use a dedicated scheme with very limited permissions, create a customer role, give that role Browse permissions only, remove the Any logged in User from every other permission.

Adding Vendors to Your Jira Runbook

Before you get started with this, you must understand the impact of allowing a vendor to access your Jira. Because by default, Jira is open to all users, inviting a customer or vendor to your Jira is going to require you to update the default permission scheme. You have to remove the Any logged in user from the Browse permission which is going to break access to every user if you don't do this correctly. Follow the subsequent steps exactly so you avoid accidentally removing Jira access for users.

First, you want to create a new permission scheme. Make a copy of your current default software scheme and if you are using "business" spaces, you'll need to apply this new permission scheme to those spaces as well.

Next, remove Any logged in user from the BROWSE permission. This step is super crucial as this is what makes Jira open to everyone and we don't want that when we have customers or vendors in our Jira. You do NOT need to remove Any logged in user from any other permissions because if you can't see the space, then the other permissions do not matter.

This is where things get tricky. Because you just removed Any logged in user, whenever you apply this new permission scheme to your space, users will lose access. You have a couple of different paths to proceed with. My personal recommendation is that you create a role for internal users and a role for vendors. You could use groups as well, but I prefer the role option because it allows each space administrator to manage who does or doesn't get access.

Once you create the Internal Users and Vendor roles, add both of those roles to the Browse Permission. The permission scheme is now ready to be applied to every space that uses the default software scheme.

Before you swap the new permission scheme, go space settings → people and add all users that should have access and grant them the "Internal User" role. For any vendors that need access, add them as well but give them the "Vendor" role. Once you do this for all the users and **for each** space, now you can swap out the permission scheme with the new one that we created. For team-managed spaces, you'll need to close those up as well. Limited or Open will not work here. The problem with all of this is that your users that typically access spaces because they are wide open will not have access problems. And there is no way of knowing 100% who all those users are.

As you can see, adding a vendor to Jira is tricky and requires some significant technical and time investments. It might be easier to user a 3rd party plugin that allows you to externally share specific work items or maybe a sync tool that syncs specific space data with a different Jira.

Making Jira Spaces Read-Only Run Book

Purpose

To temporarily or permanently restrict changes to a Jira space while still allowing users to view work itemss. This is often required when:

- A space is archived but not moved to Jira's built-in Archive (Data Center or Premium/Enterprise Cloud feature)
- Audits or compliance checks are in progress
- Migration is happening and space data must be preserved in its current state
- The space is being phased out but needs to remain visible

Prerequisites

- Jira Administrator permissions (for permission schemes and space settings)
- Knowledge of whether the space is Company-managed or Team-managed
- Agreement with stakeholders on:
  - Duration of read-only mode
  - Who, if anyone, will still have edit permissions

Steps for Team-Managed Spaces

(Team-managed spaces don't use global permission schemes — control is done via Roles)

1. Go to space Settings → Access
Review roles and permissions for all groups.

2. Edit Roles
For all user roles except Admin:

- Remove permissions for:
  - Editing work items
  - Creating work items
  - Commenting (optional)
  - Transitioning work items

Admins can retain edit access if needed.

Steps for Company-Managed spaces

1. Identify Current Permission Scheme

- Navigate to:
  space settings → Permissions
- Note the name of the permission scheme in use.

2. Create a Read-Only Permission Scheme

- Go to:
  Jira Settings → Work Items → Permission schemes
- Click Copy next to the current scheme to create a duplicate.
- Rename it something like:
  Read-Only - <spaceName>

3. Remove All Edit/Transition Permissions

In the copied scheme, remove permissions such as:

- Edit work items
- Transition work items
- Add Comments (optional — if you want comments blocked)
- Delete work items
- Create work items
- Attach Files
- Link work items (optional)
- Assign work items / Schedule work items (optional)
- Work On work items

Keep only: Browse spaces, View Voters and Watchers, View Development Tools, and other "view" permissions.

4. Assign the Read-Only Scheme to the space

- Go to the space's Permission scheme setting.
- Assign the new read-only scheme.

Top 5 Automation Rules for Everyone

Auto-Assign Work

- **Trigger:** Work item created
- **Condition:** Check work item type (e.g., Bug, Story, Task) or component
- **Action:** Assign to a default assignee or round-robin between team members (using groups)

** Ensures new work isn't left unassigned.

Send Reminders for Stale Work Items

- **Trigger:** Scheduled (e.g., run daily at 9 AM)
- **Condition:** Work items in "In Progress" for > 7 days without update
- **Action:** Add comment "This work item hasn't been updated in a week, please review" or notify assignee via Slack/Email

** Keeps work from getting stuck and forgotten.

Auto-Close Work Items if No Response from Reporter

- **Use Case:** Keeps backlog clean, especially in Service or Support spaces.
- **Trigger:** Scheduled (e.g., every day at 6 AM).
- **Condition:** Work item with status = "Waiting for Customer" AND no update for X days.
- **Action:** Transition work item to "Closed" and add a comment: "We haven't heard back in 5 days, so we're closing this work item. Please reopen if you still need help."

** Reduces clutter and keeps focus on active requests.

Copy Labels or Components or any field from Epic to Child Work Items

- **Use Case:** Ensures Stories/Tasks automatically inherit metadata from their Epic.
- **Trigger:** Work Item created or linked to an Epic.
- **Condition:** If work type = Story, Task, or Bug AND Epic field is not empty.
- **Action:** Copy values from Parent → Labels, Components, or Custom Fields.

** Helps with reporting at Epic level and saves users from manual updates.

Notify Slack/Teams Channels on Key Transitions

- **Use Case:** Keeps the team updated in real time without refreshing Jira.
- **Trigger:** Work Item transitioned (e.g., to "In Review," "Ready for Release," or "Done").
- **Condition:** Only for certain work types (e.g., Bugs or High Priority Work).
- **Action:** Send message to Slack/Teams channel with details: "ISSUE-123 (Bug) moved to Done by Alex."

** Improves transparency and communication as team doesn't have to context switch.

JQL Queries

Find the work I have in current sprint
space = [project_name] and assignee = currentUser() and sprint in openSprints()

Find any bug that is assigned to me that aren't fixed
assignee = current(user() and issuetype = Bug and resolution = unresolved

Find work that currently in a sprint, but not assigned to anyone
sprint in openSprints() and assignee is empty

Find work that in code review and assigned to me
status = "Code Review" and assignee = currentUser()

Find every Epic that is currently In Progress
issuetype = Epic and statusCategory = "In Progress"

Find every Epic that isn't complete
issuetype = Epic and statusCategory in ("To-do", "In Progress")

Find every work item that is blocked
"Flagged[Checkboxes]" is not empty

Find every work item that has been completed within the last month
statusCategory = Done and resolved >= startOfMonth(-1) and resolved <= endOfMonth(-1)

Find overdue work that isn't already completed
duedate < now() and resolution = Unresolved

Find Unestimated work
issuetype in (standardWorkTypes()) and "Story Points[NUMBER]" is EMPTY

Find Blocked items by blocked relationship
issueLinkType = "is blocked by" and resolution = Unresolved

Find work that is in upcoming release
fixVersion in earliestUnreleasedVersion() and resolution = Unresolved

Find work that is super high priority and isn't complete yet
statusCategory != Done and priority in (Critical, Highest, High)

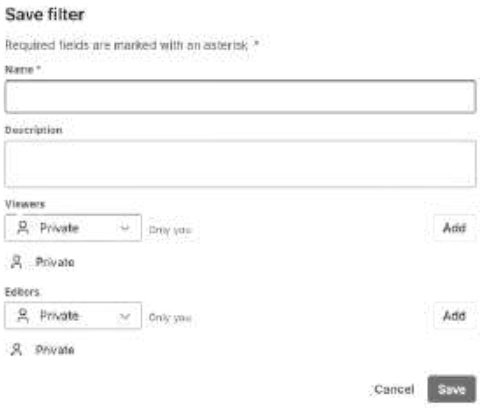
Find all unasssigned Bugs
issuetype = Bug and assignee = empty

How to Create and Share Filters with Your Team

Out of the box, your filters are configured to be private. This presents a big problem as folks normally want to share their filters and when they share the link to their filter, they usually run into a problem because their filter is set to private.

There is a global setting that your Jira admin can toggle to not make filters automatically private, but chances are you don't have the right admin rights to do this, so instead, I'll show you what you need to do to make your filters shareable.

First, create a filter. When you click on Save Filter, you'll be prompted to provide a name and description. Right below that, you'll see the "viewers" and "editors' section. To facilitate sharing a filter, you'll want to focus on the "viewers" section.



You have a few different options when settings the "viewers"

At a minimum, you should set your viewers to space and select the relevant space(s). This will allow you to share your filter with anyone that has access to your space.

If you need to share your filter with a director or executive, they will most likely not have access to a specific space, so you should set the viewers to My organization. This will open up your possibilities to share to every licensed user which isn't ideal, but will ensure that your executives and directors can access the filter data. Keep in mind that space Access still governs if someone will be able to see the work items or not.

If you know the name of your Jira group and the users within that group, then Group is a good option. If you have no idea what groups are available or who is in the groups, then use the space instead.

Finally, you have the option to specifically add a user. This is okay if you only want to share the filter with a specific person, but if you have to share with many, the space is still going to be a better option.

Global Jira Permission Settings Tips

There are a few global permissions that shouldn't really be tweaked often, but it is important for you to understand what these permissions do. There is one permission in particular, the ability to create team-managed projects that is set to public by default. You'll want to turn this "feature" off. Additionally, a common problem that users have is that they can't tag/assign work to someone on their team. After you've exhausted licensing and space permission troubleshooting, you should confirm that the user is in the Browse Users and Groups permissions below. Some apps might add their own global permissions (not noted here).

Access Global Permissions: Gear → System → Global permissions under Security

Permission	Definition	Tip
Administer Jira	Create and administer spaces, work types, fields, workflows, and schemes for all spaces. Users with this permission can perform most administration tasks, except: managing users, importing data, and editing system email settings.	You should have the smallest number of groups here. This empowers the out of the box groups that can administer Jira
Browse users and groups	View and select users or groups from the user picker, and share work items. Users with this permission can see the names of all users and groups on your site.	This allows users to tag someone in the comments or assign a work item to someone. If they aren't in this permission, they wont be able to search for users.
Share dashboards and filters	Share dashboards and filters with other users.	This allows users to share the dashboards or filters they create. Doesn't impact ability to create, just share.
Manage group filter subscriptions	Create and delete group filter subscriptions.	Allows users to create subscriptions which are filter results that get emailed.
Make bulk changes	Modify collections of work items at once. For example, resolve multiple work items in one step.	Allows the users to update work items in bulk
Atlassian Home for Jira Cloud space connection issue guidance	Enables the Atlassian Home for Jira Cloud issue glance field in issue sidebars, for connecting a Jira issue to a space overview	This is a new feature and you may want to turn off if you aren't using Atlassian Home
Create team-managed spaces	Create spaces separate from shared configurations and schemes. Team-managed spaces don't affect existing spaces or shared configurations like workflows, fields or permissions. Only licensed users can create team-managed spaces.	Remove the ability for anyone to create team-managed spaces. This should be updated to just your Jira Admin group(s)

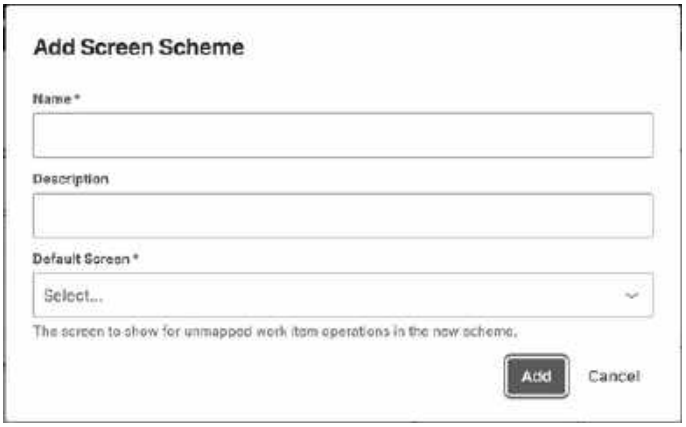
Jira Product Settings

Global, product specific settings that you should review and ensure are appropriately configured for your team. Some should be turned off, but keep in mind that turning them off (or on) affects all your users.

Configuration	Definition	Tip
Parallel Sprints	Allow multiple sprints to run at the same time. This feature doesn't apply to team-managed spaces.	Turned on by default, but you should consider turning off since this is a global setting and most teams shouldn't be running parallel sprints
Store Jira data on your own device	Store frequently accessed data on your device to help your Jira board and backlog load faster. This setting applies to all users on your Jira site. When the setting is turned off, any stored data will be deleted	Turned on by default, consider turning off if you have strict data residency requirements.
Public access for forms	Allow Jira users to make forms available to anyone online who has the link. When this setting is turned off, public access will be disabled for new forms, and any existing forms with public access will be deactivated.	Similar to how JSM's portal works, this enables Jira users to share Jira forms and anyone can submit, regardless of Jira license.
Storage management	Displays Work Items taking up a lot of space (attachments)	Allows you to view which work items have large attachments but can't actually do anything from this menu. You need to go to the work item to delete large attachments
Application Links	Application links let you integrate Jira with another Atlassian product or external application so they can exchange information, resources, and functionalities.	Connect Jira with another Jira. Useful when you have multiple cloud or DC Jira's that need to share data.
Compass	Control how Compass integrates with Jira.	Configures Jira spaces to leverage Compass components. You should revert back to Jira components if your team doesn't use Compass.
DVCS accounts	Enjoy the seamless integration of work items and code when you connect Jira to your Bitbucket Cloud workspace.	Connect to your Bitbucket.

Adding a New Screen for a Specific Work Type

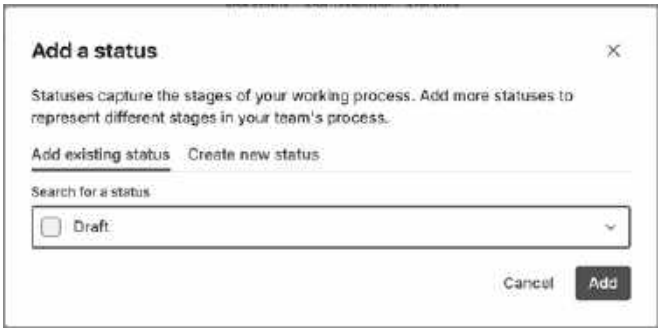
1. Click on the gear → Work Items
2. Click on Screens (on the left menu)
3. While you might be tempted to create a blank screen, I recommend you use an existing screen from an existing Jira space and make a copy
    i. Search for the screen → Click (...) → copy
    ii. Name the copy something valuable and clear
4. Search for your newly created Screen
5. Click on the screen name of your new Screen
    i. Modify (Add, remove, reorder your fields as need)
6. Click on Screen Schemes (on the left menu) → Add Screen Scheme (blue button top right)
7. Give it a name and description
    i. Under Screen, select our new screen from the previous step → Add



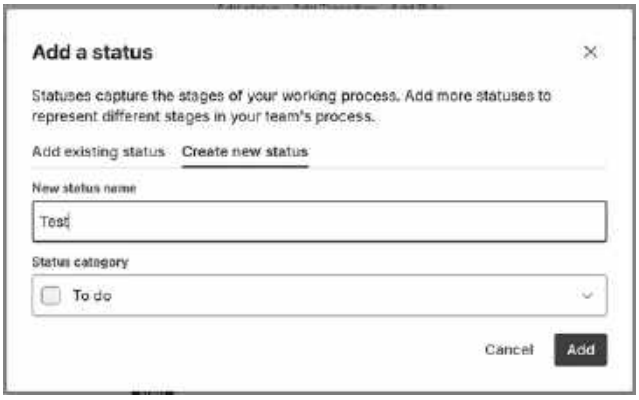
8. Click on Work type screen scheme (on the left menu)
9. Find the Work Type Screen Scheme for your space
10. Click the Work Type Screen Scheme → Click on Associate a worktype with a Screen Scheme (light gray button, top right)
11. Select the WorkType → The new screen scheme we made earlier → Click Add

Adding a New Status to an Existing Workflow

1. Click on the gear → Work Items
2. Scroll down on the left and click on Workflows
3. Identify the Workflow you want to add a status to and click the edit button
4. This will take you to the workflow you want to edit
5. Click Add Status
    - If using an existing status, it will show up when you start typing. If you are using the new editor, there is a section to search for existing statuses. Click Add.



    - If creating a new status, type in whatever name you want. If using the new workflow editor, there is a section to create a completely new status.
        - → click (new status)
        - → pick the preferred status category
            - To do = no work is being done,
            - In progress = work is actively being performed, and
            - done = work is completed)
        - Click on Add.

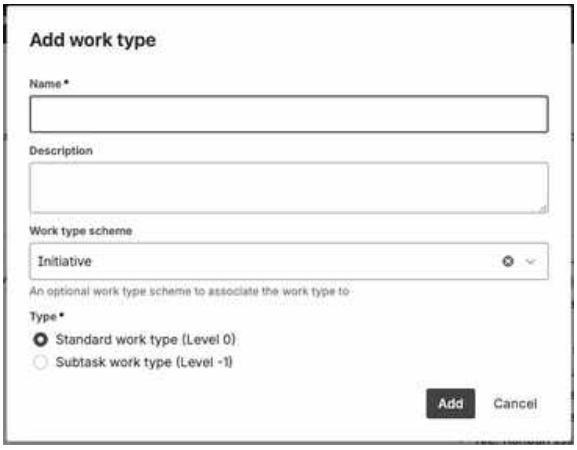


    - When finished adding statuses, Update workflow and don't worry about having to save a copy.

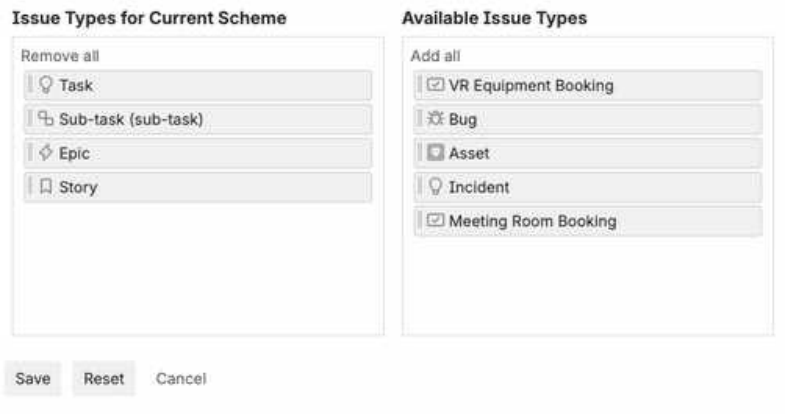
Adding New Work Type to a Jira Space

If this is for a new work type that doesn't exist:

1. Click on gear → Work Items, this will automatically land you in the location to add new types
2. Click on Add work type (blue button, top right)
3. Give it a name and description.
4. Select the appropriate Work type scheme if you have one.
5. Select Standard work type (Level 0)



6. Click Add
7. Go back to your newly created work type
8. Find the (...) on the right side → Edit
9. Change the Avatar (otherwise, all work types will look the same). Then click Update.
10. On the left side menu, click on Work types schemes
11. Click the (...) next to the scheme you want to add the work type → Edit
12. Pull the Work Type from the Available Issue Types (Atlassian is working on updating the word “Issue Types” to “Work Types”) add it to Issue Types for Current Scheme

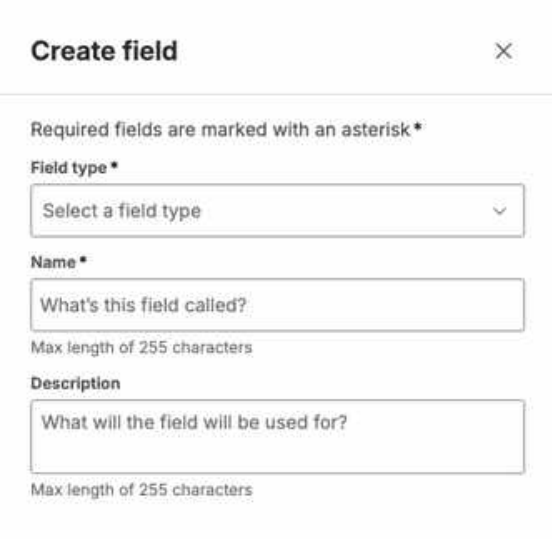


13. Click Save

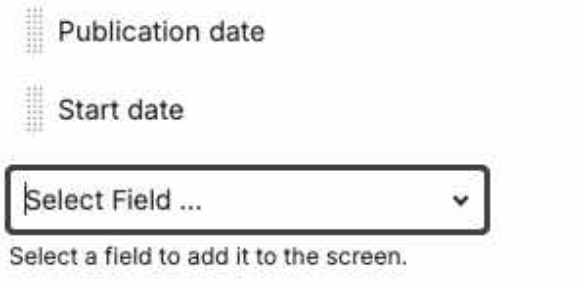
*This impacts/applies to all spaces that share this work-type scheme

Creating and Adding a Custom Field to a Screen

1. Click on gear → Work Items → Fields
2. Click on Create new field
3. Pick a Field type from the drop-down menu
4. Give your Custom field a name. Depending on the type of field you create, you may need to provide values as well.



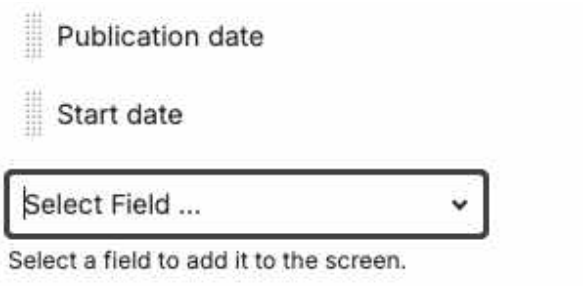
5. Click Create
6. Click Screens on the left side menu
7. Click on the name of the screen you want to add the field to
8. Go all the way to the bottom, where you will see a dropdown and find your screen



9. Once your field is in the screen list, that's it. No need to save anything.

Adding an Existing Field to an Existing Screen

1. Go to Space setting → Work items → Screens
2. Click Screens on the left side menu
3. Click on the name of the screen you want to add the field to
4. Go all the way to the bottom, where you will see a dropdown and find your screen



5. Once your field is in the screen list, that's it. No need to save anything.

Deleting a Status, Transition, or Adding Transitions

Know the workflow name you want to update first

1. Click on the gear → Work items → Workflows
2. Find the workflow you want to remove a status from and click on (...) → copy

3a. Delete any status or transition

3b. Add a new status - See instructions for “Adding a New Status to an Existing Workflow” page 43

3c. Add a transition by connecting 2 statuses together

i) Define the transition name (verb)

ii) Make sure you account for all the transitions (in and out of each status)

iii) Most only create a "happy" path, but you need transitions to move backwards in your workflow

iv) When you are finished adding transitions, click the checkbox that says "Show transition labels" (top left)

4. If everything looks good, you do NOT need to publish; it's a more complicated process
5. In a new tab go to the space that needs the new workflow
6. Go to the space settings → Workflows → Add workflow → Add existing

*if your workflow scheme is shared by multiple projects, you'll need to add the new workflow to the workflow scheme instead

7. Find your "New" workflow → Next
8. Select the work type(s) that will be associated with your new workflow → Finish
9. Go to your tab with the workflow and **Publish** your workflow

How to add and organize your users?

1. Create the following roles:
    - Developer: r/w
    - User: r
2. Create the following groups:
    - Developers • Users
3. In the permissions, scheme modify the browser permission Add:
    - Role: Developers
    - Role: Users
    - Group: Site Admins
    - Remove: Any logged-in user

3.1 Create, Move, Transition, Edit

- Remove: Any logged-in users
- Add: role: Developers
- Add: role: Administrators

4. In the People Setting of your Jira Project add:
    - Group Developers to role Developer
    - Group Users to the role User
    - Add Yourself to the role Administrator
5. Invite/Add your users to the appropriate group depending on if you want them to be an admin, developer, or user.

Space Settings Guide

Details:
- Change the name of your space and key
- The URL doesn't do anything helpful, you can leave that alone
- Space type is locked and determined when the space created
- Space category helps organize if you have a lot of Spaces
- You can upload a custom avatar or choose from available options
- You can provide a description of your spaces
- The space lead "owns" the space
- Leave the default assignee as unassigned

Summary:
- Shows you all the configurations for the space at a glance

People:
- This is where you add/grant users access to your space
- Can grant access to individuals or groups
- Permissions:
- Configured by Jira Administrator and controls what users are allowed to do in the space

Notifications:
- Determines what events trigger a notification and who receives it
- 1 notification scheme is typically used for all spaces or your admin can make you a new one

Automations:
- Automate tedious tasks such as moving a work item to In Progress when a branch is created

Features:
- All planning features are automatically enabled by default
- Disable/Enable development, operations, and Confluence features

Workflows:
- Configure the available statuses in your project
- Only Jira admins can make modifications
- By default, all work types share the same workflow
- Each work type can hare its own workflow

Work Types:
- Configure which work types can be created/used in your space

Work Type Layout:
- Configure where and how fields are displayed when a user is viewing a work item

Work Type Screens:
- Configure which fields are available in your space

Work Type Fields:
- Configure which fields are required upon creation of a work item

Work Type Collectors:
- Allow anyone on the internet to create work items in your project
- Not recommended

Work Type Security:
- Configure which users can see a work item
- Just because a user has access to a space, doesn't mean they need to see all work items
- Great for when you have contractors/external resources

Component:
- Define the values that are available in the component fields

Apps:
- Configure plugins that are available for your project

Development:
- Connect to your team's source code repository

Kanban Board Guide

- Default Statuses: Backlog → Selected for Development → In Progress → Done
- All W types will show up on the Kanban board
- A "dedicated" backlog can be enabled to hide non-prioritized work
- A dedicated backlog allows the team to select items intended for "development" properly
- If Epics panel is enabled, Epics will disappear from the board/backlog
- Subtasks are visible (unlike in a scrum board) and grouped under their respective parent
- WIP limits can be set on each desired column
- Completed work items will remain in the farthest-most right column until work items are either released OR after 1, 2, 4 weeks
- You cannot estimate or view story points in a Kanban board
- When a log is enabled, moving an issue from the status of "Backlog" to the status of "Selected for Development" will then allow for the issue to show up in the Kanban board
- Enabling a backlog will help keep your Kanban board clean and focused

Board Settings Guide

General:
- Change the board name
- Define any additional board admins (can make changes to the board settings)
- Change the location of the board
- Edit the filter query when you want to modify the work items that show up in the board
- By default, the board will show all unresolved work items in a space
- You can swap the saved filter for another filter (if you don't want to change the existing filter)
- Control space shares as it determines who can see the board
- Filter query shows you the criteria for which work items will show up
- By default, the board has rank enable
- If you change the ORDER BY in the board filter, you will lose the ability to rank
- Based on your filter, the board may include work items from different spaces (brakes basic roadmap)

Column:
- Column constraints will add limits on how many work items can be in a column
- Make sure you are always using a simplified workflow otherwise, your work items may not resolve properly
- Add columns and statuses as desired
- Remove unwanted statuses and columns by adding to unmapped section

Swimlanes:
- Change how work items are grouped/viewed on the board

Quick Filters:
- Create custom JQL queries that allow you to quickly filter through the work items in your board

Card Color:
- Add a super tiny sliver of color to the left side of a card based on specific criteria

Card Layout:
- Add up to 3 fields to be displayed within the body of a work item's card

Estimation:
- Determine if estimates are captured as Story points or using hours (original estimates)

Work Days:
- Define which days your team works in order for board reports to be more accurate

Work Item Detail View:
- Configure how fields show up when viewing and work items

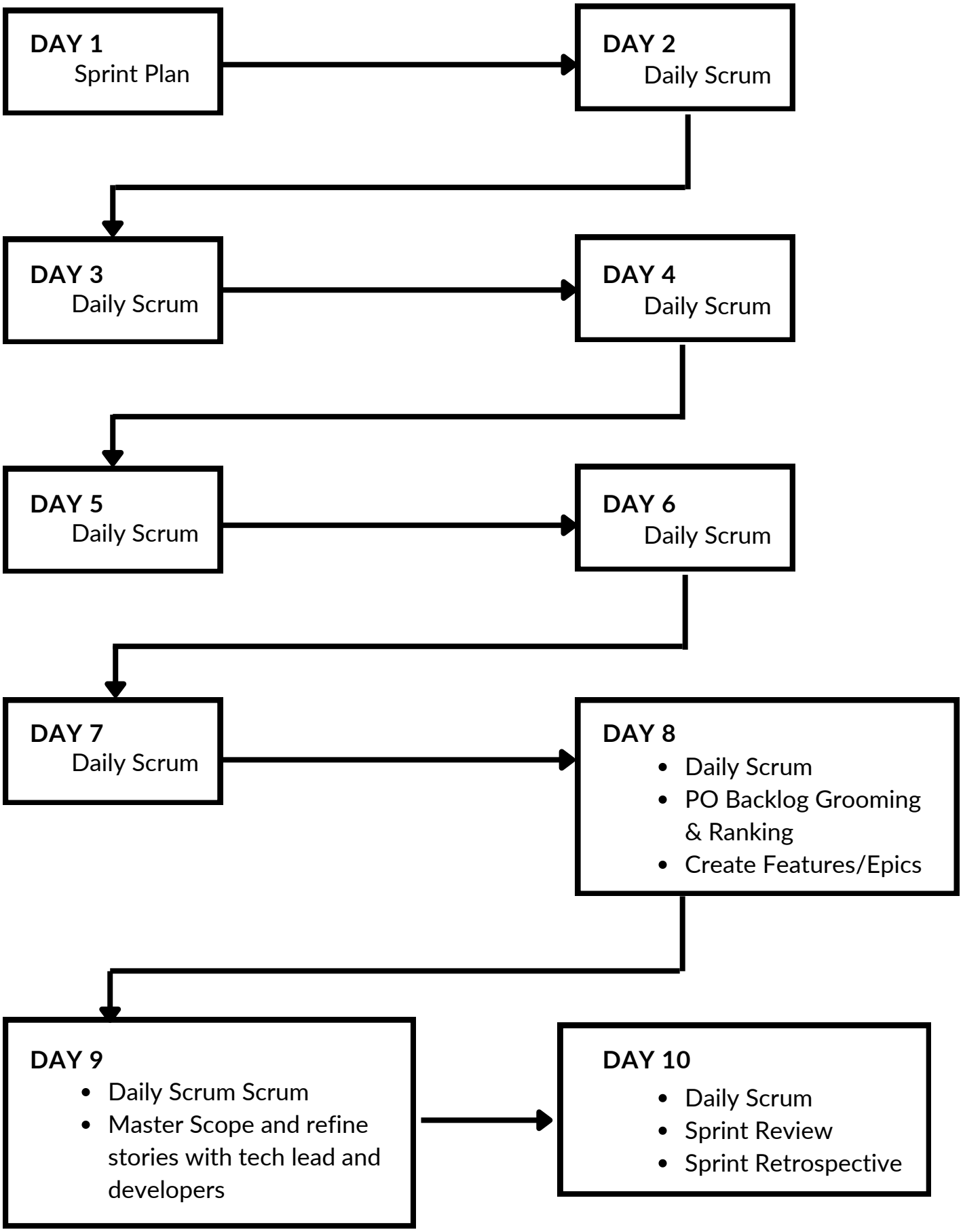
Roadmap:
- Enable/Disable the basic "free" roadmap
- Enable/Disable if Sprints are displayed on the roadmap when a story is planed into a Sprint

Insights:
- Enable/Disable which insights are presented in the backlog and/or board view

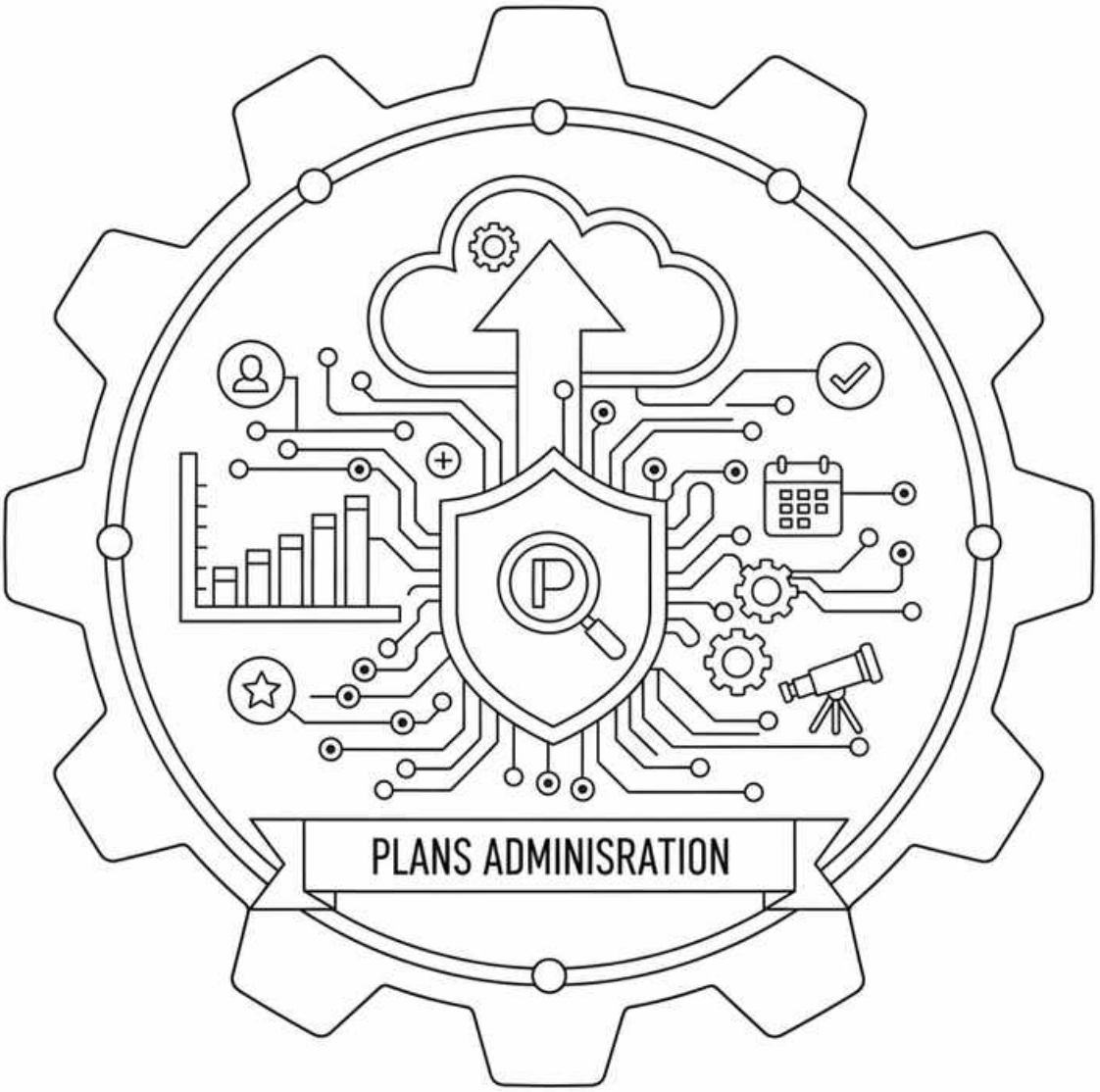
Scrum Board Guide

- Default Status: To do → In Progress → Done
- Only Stories, Tasks, and Bugs will show up
- Epics are only visible in the Epic panel
- Stories can be associated to an Epic by dragging the Story to an Epic in the Epic panel (or using parent)
- Story points can be added while a work item is in the Backlog or in a Sprint
- When a work item is moved to a Sprint, the work item is not transitioned (there is no "Backlog" status in a Scrum board)
- A work item is in a Sprint whenever the Sprint field is set
- You know a Sprint is active because you see the story points in the backlog and your active Sprint will NOT be empty
- Setting dates on your Sprint doesn't actually do anything
    - when you physically click the Start Sprint button, it sets the Sprint Start Date/Time
    - when you physically click the Complete Sprint button, it sets the Sprint End Date/Time
- If you forget to click either button your Sprint will either not start, or, it will remain active indefinitely
- By default, your Backlog can be ranked, but you can change this by altering your board filter to include ORDER BY Any field
- If you assign work items to individuals, you'll then be able to see how many work items are assigned to each individual and how many story points they have for that Sprint. This is the closes thing Jira does for Capacity Management.

The Sprint Cadence Guide



GETTING STARTED WITH PLANS ADMINISTRATION



DAM GOOD BOOKING

APWIDE BOOKING

Jira Plan Settings

Details:

Plan Name	Name of the Plan, pick something big and descriptive and not specific to a board or Jira space/space.
Plan Lead	The owner of the plan, typically the space Manager
Who Can Edit The Plan	This should be set to restricted if you don't want everyone making changes otherwise, anyone can edit the data in the plan
Who Can View The Plan	This can be left open since ultimately the Jira space permissions will control who can/can't view the Jira data.

Estimation and Scheduling:

Estimate Work Using	Stories - Use this if your team is using Story Points. Days/Hours - Use this if your team uses Original Estimate, then just pick your preference between Days or Hours
Start Date	Any date field can be used here.
End Date	Any date field can be used here.
Work Items with Empty Start or End Dates	Defines what dates default to a work item when the start or end date are missing.
Dependent Work Items be scheduled to the same iteration	If selected, any work item that is linked together will be scheduled to the same sprint.

Work in your plan:

Overview	Visually confirm which work items are included in your plan. Edit plan scope to add additional Spaces, Boards, or Filters.
Removed Work Items	Check this setting if you ever lose work items. By default, after an item has been closed for 30 days, it will disappear from your plan. Update the date to at least 100 days if doing quarterly planning. Just be careful with hitting the 5000 work item limit of the plan.

Timeline:

Saved Views	Review which views exist in your Plan. Review who created the view, who was the last person to update, and when the view was last updated. You can also remove the view altogether or make a specific view, the default view.
Fields Added to Timeline	Review which custom fields have been added to the plan. You can review the description of the field (if available), the type, and the work items the field applies to. You can also remove the field from this setting view.

Program Boards:

Active Boards	Shows your active programs boards. Can have 10 per plan. Admins have the ability to delete the program board.
Past Boards	Review previous program boards.

Features:

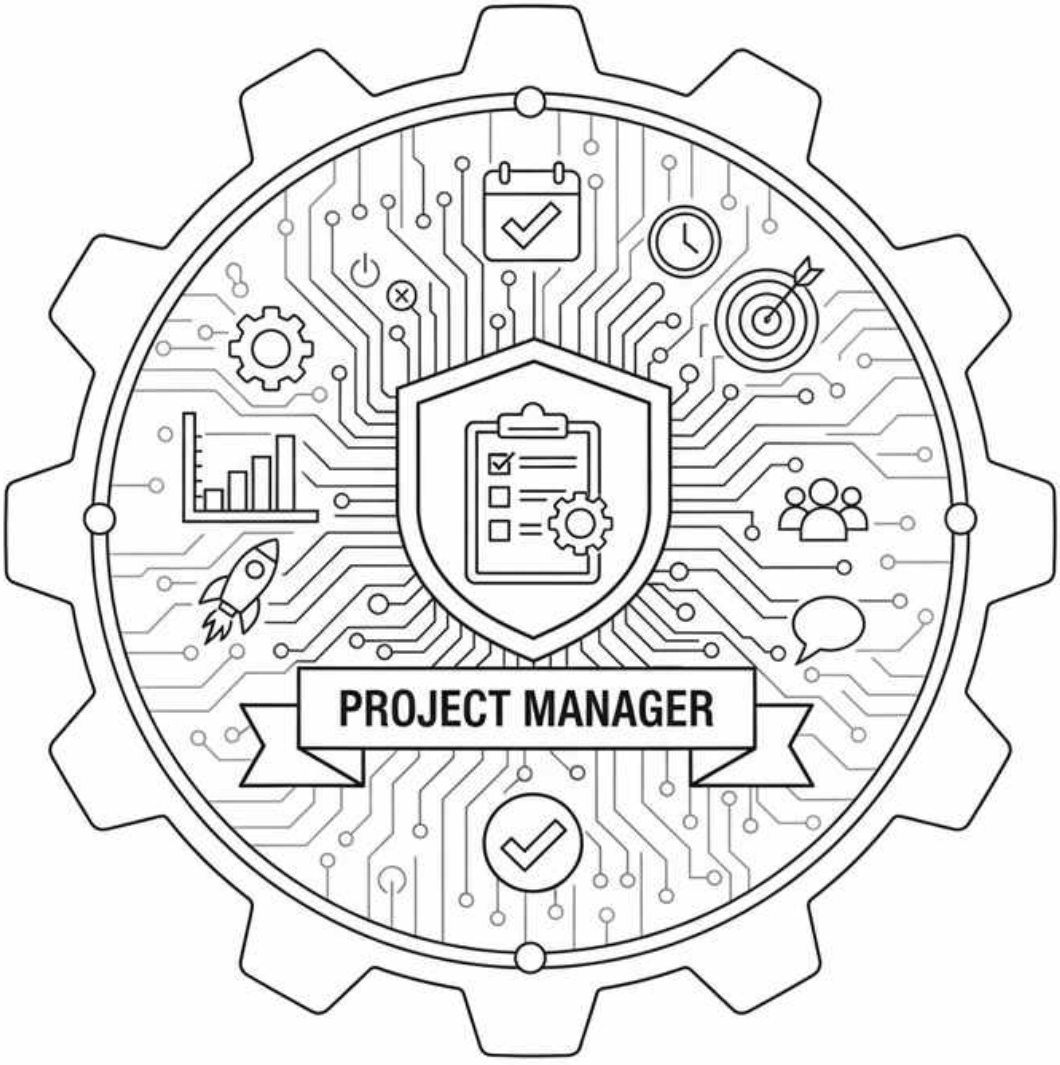
Scenarios	Create multiple versions of your plan to tweak resources, dates, and other important decision making data.
Auto-scheduler	Automatically schedule work items that have been estimated in your plan, then adjust the generated plan to meet your exact needs.
Releases	Schedule work using releases for agile software teams, or hide them for a simpler experience.

Jira Advanced Plan Settings

These are global Plan settings that affect all your Plans. Carefully change these configurations as they are not Plan specific, but rather global.

Setting	Definition	Tip
Permissions	These permissions govern all plans. They specify the groups that can potentially be administrators, users or viewers of the Plans.	Make sure as many licensed Jira users have access as plan viewers. Especially ensure your executives can at least view the plans.
Hierarchy	Manage the work type hierarchy by changing the level names and structuring the hierarchy to your needs.	Add work item types above the Epic or rename the Epic. You cannot add work item types below the Epic.
Dependencies	Select the work item link types that will be treated as dependencies in plans	By default, Jira sets dependencies as Blocked/Blocked By. Here, you can add more relationship types.
Trash	Deleted Plans show up here	Permanently delete or restore Plans.
Archive	Archived Plans show up here	Restore archived Plans.
Financial Year	Configure the start month of the financial year	This financial year sets the quarters for all the Plans

GETTING STARTED WITH PLANS FOR PROJECT MANAGERS



Creating Your First Plan Tips and Tricks

Plans are a Jira Premium/Enterprise feature that every team should be using. Plans give you a birds eye view to your Projects and Initiatives that your various teams are working on. Instead of being limited by individual Jira spaces, you can aggregate all your Jira spaces in a single place, a Jira Plan! The following is a guide to getting started with Jira Plans.

First, make sure you are on Jira Premium/Enterprise or DC. Jira Plans are not available for Free or Standard cloud subscribers. Next, click on Plans and create a new Plan.

You Jira admin might have configured global settings preventing you from creating a plan. At that point, you will want to speak with your Jira admin to either give you permission or help you create your Plan.

Assuming you can create a Plan, let's proceed. Your Plan will need a name. This is where a lot of people make their first mistake. Plans can be used by all types of teams. Big teams, small teams, geographically dispersed teams, all teams can take advantage of Plans. But, where Plans provides the biggest return on investment, its when you combine two or more Jira spaces together. You Jira Plans can present you with up to 5000 work items! With all of that said, pick a name for your Plan that doesn't just represent the small work your team is working on, but rather think bigger. The name of your plan should be something big. Think product, major project, or major initiative that a collection of teams are working towards delivering.

With respect to access, I recommend you leave Open as restricted your plan doesn't make too much sense. Because of how Plans and Jira work, anyone that has access to the work items within their Jira space will be able to modify the data. Making your plan private doesn't make the data private. Therefore, allow your stakeholders and team to access the space freely, but restrict who can edit your views in the plan which you do in the Plan settings after the Plan has been created.

Finally, add your work items to the Plan. Remember, you are limited to 5000 work items and Jira doesn't tell you if you are getting close to 5000 work items until after you hit the limit. My recommendation is add a space or two and then, after your plan is created, add more spaces/boards/filters because in the Plan settings, Atlassian will show you how many work items are in your Plan.

Last tip, pull in your space if you are getting started with Plans for the first time. Boards are great as they enable more advanced features, but sometimes boards have filters that limit the data and you end up missing data in your plan. I avoid pulling in filters for the same reason, they limit data that is retrieved and I've learned that most folks tend to miss data by using Boards and Filters.

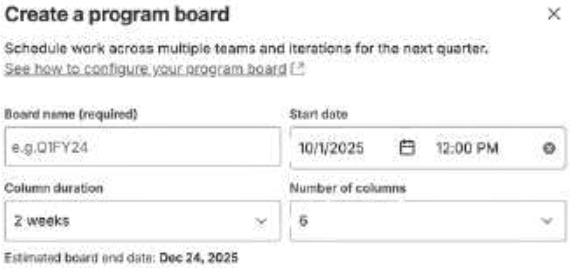
PI Planning with Jira Plans Guide

This is an advanced feature that can be used by all teams, but folks wanting to do Program Increment (PI) planning will benefit the most. To use this feature, your team needs to be using a Scrum Board and be using Story Points as their estimation method. When you create your plan, you also need to use Boards as your resource instead of Spaces or Filters. Finally, you'll need to create at least 6 sprints to fully take advantage of the Program board

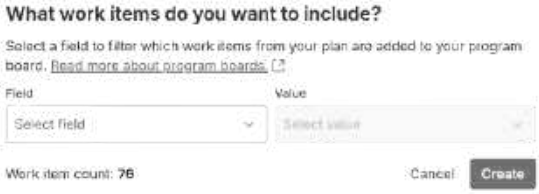
Create a Plan with Boards as your source then click on Program within the Plan's UI

Click on Create program board

Provide a Board name and pick the start date for your program increment.
Select the duration of each iteration (sprint) and determine how many iterations you'll have



Select the work that you want to include in your Program. If you don't put anything in, all the items in your board will be included, so only fill out this section if you want to intentionally include or remove work items.



Now, all you need to do is drag work items from the left and place in their respective iteration/sprint that your team should work on them. You can add dependencies to show how all the work is related just like you would in a physical program board. Once you are done, save your changes and all the work will automatically be added to their respective sprints.

Cross-Team Dependency Management With Jira Plans Tips

One of the best features of Jira Plans is that you can visually see how all your work is connected. Unliked Jira spaces that do a really good job of displaying all the work within the space, Plans take it to the next level by showing you all the work across all the data sources you pull in to your Plan.

When you create your plan, make sure you include more than one source. If you simply include a single Space/Board/Filter, then this will defeat the purpose. Make sure you include at least 2 spaces that you have access to.

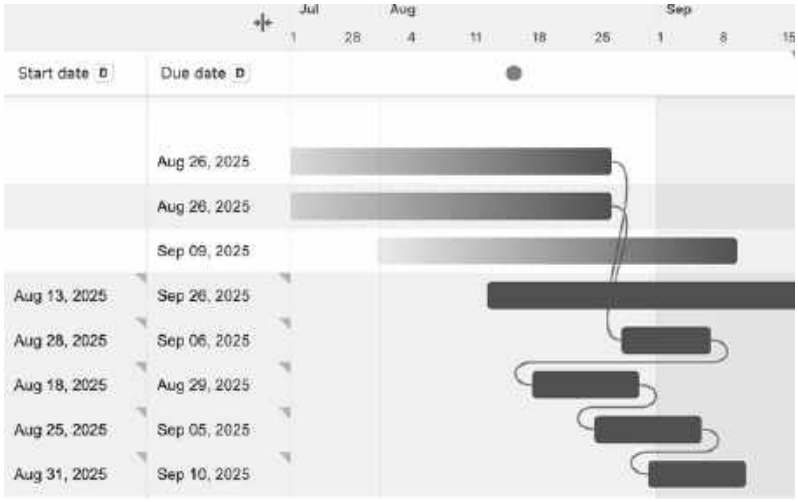
While you can add a work item link directly to the work item, in order to visualize the relationship, you need to provide a start and an end date. Without these dates, you won't be able to truly visualize the dependencies. Also, you may want to consider making sure your dependencies are visible by "line" and not by "badge", but this is a personal preference based on the number of dependencies that you have.

You should also keep in mind that dependencies can be across any type of work. You can link Epic to Epic, but you can also do Epic to Story or even Epic to Sub-task. That last one might not make a whole lot of sense, but if it works for you, do it!

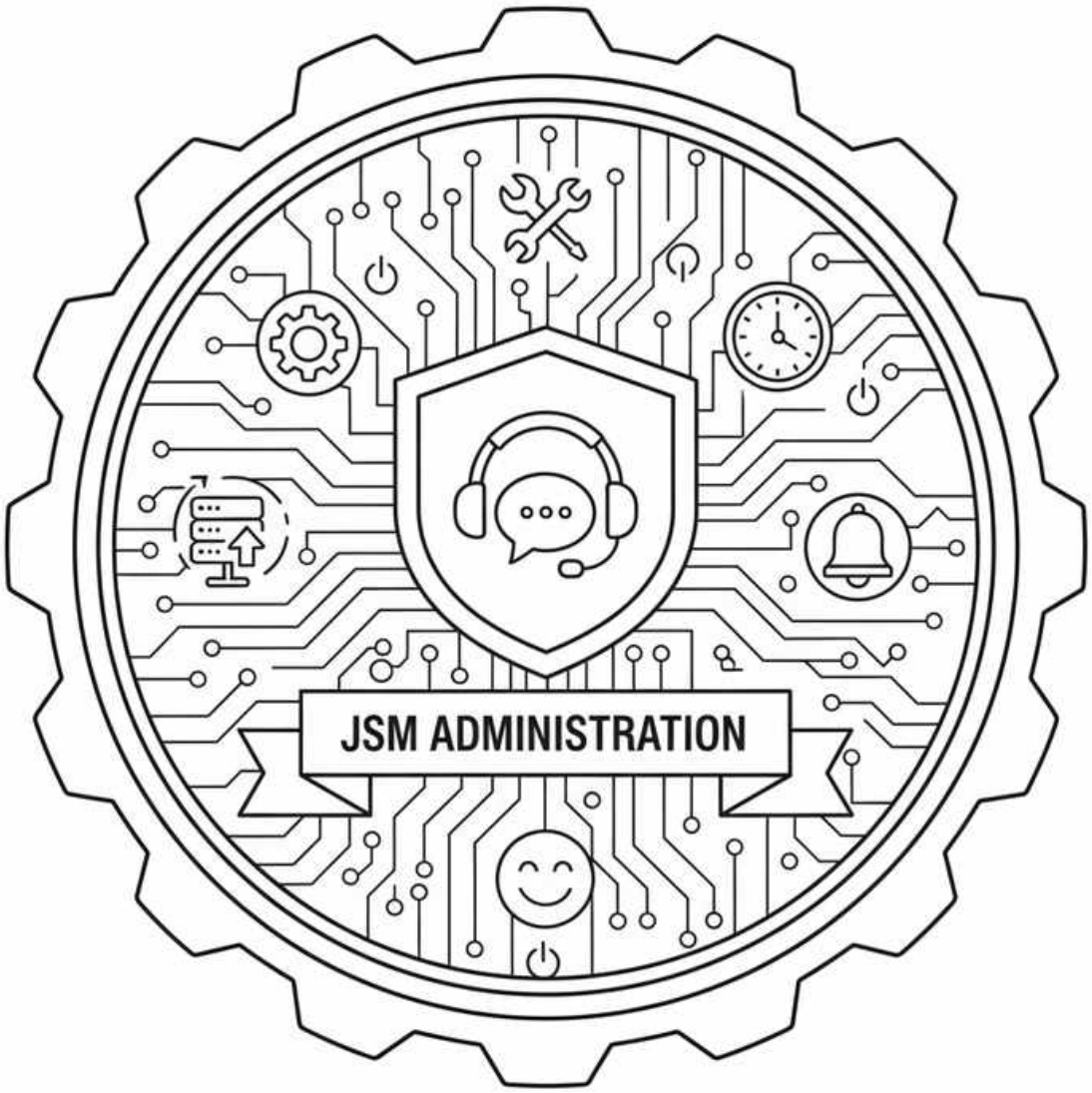
Out of the box, you can only link work items with the Blocks and Is Blocked By relationship, but your Jira administrator can add additional relationships.

Red dependencies lines mean that the end date happens after the start date of the task that is dependent. This is not good. You want the end date to happen before or on the start of the next item.

Finally, to visually see the dependencies, make sure your in the Timeline view. If not, you will not be able to see the dates or the dependencies.



GETTING STARTED WITH JSM ADMINISTRATION



Determining JSM Project Template Guide

Like with Jira, JSM ships with many different space templates based on your specific needs. This guide is going to go over each template and help you figure out which space template is right for the job. With the templates, there are basically two ways to go about this. You can start with a blank slate and build everything on your own, or you can start with a pre-configured template and modify some or most of the settings. I personally prefer to start with a blank slate, but choose whatever makes sense to you. I would recommend you use each template so you can have better data to determine which route is best for you.

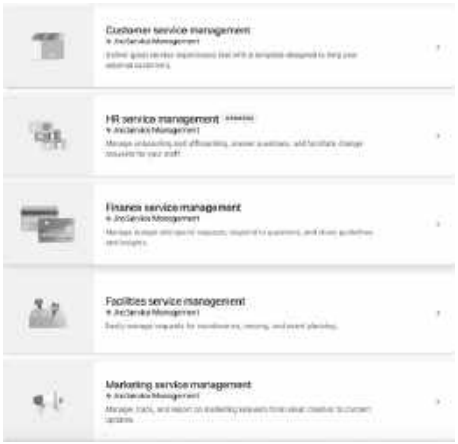
The blank template gives you the basic barebones configurations and you are expected to know how to modify all the settings such as the workflows, work item types, request types, etc.

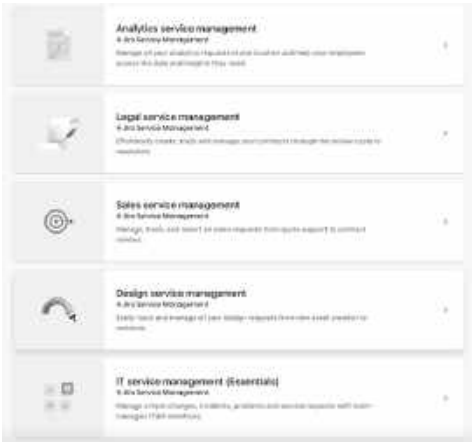
If you aren't a Premium/Enterprise subscriber or you don't want all the bells and whistles of the ITSM template, but need more than the General service management, the Basic ITSM template might be for you. You get service requests and incidents out of the box. The template is the best of both worlds.

The ITSM templates gives you all the bells and whistles of JSM, but it's only available to Premium/Enterprise subscribers. It has everything you could possibly need to run a full ITSM solution for your company and probably even more. Choose this only if you really need all those extra features.

This is my favorite template. It gives you just enough configuration to get started, but doesn't overwhelm you like the ITSM template. When in doubt, use this template.

The rest of the templates are very use case specific. If you are setting up JSM for one of these teams, these templates will probably serve as a good starting point. I would recommend you try them out and determine if they actually help or if one of the more generic templates from above are a better fit.





How to determine what questions to ask stakeholders before creating a JSM Space?

Now that you know the different space types and templates available, the most important question is how do you determine which template you need.

The first step is to determine how much “help” you want from the out of the box templates. Nine times out of ten, I usually select the General service management template. For those teams that want every single feature and are on JSM Premium/Enterprise, then the ITSM template might make more sense.

Each template is going to come pre-packaged with work item types, workflows, and more importantly, request types. My recommendation is check the details of each template and see if one of the templates will work for you and your team. If you feel overwhelmed by all the Basic and ITSM template, I would recommend using the General service management template as that one gives you just enough to get started, but doesn’t overwhelm you.

Which template you select is ultimately going to come down to the request types that are available for each space template. Therefore, you should have some idea of what types of requests you want to make available for your customers.

What is important for you to understand and remember is that request types are ultimately tied to work item types. And because of this, that means that request types are ultimately tied to the workflow that belongs to the work item type. Therefore, my recommendation is to figure out how many “unique” processes you have to help you determine how many request types you need. With the ITSM and Basic ITSM space templates, you are going to get 2+ work item types and dozens of request types. But the key piece here is you have 2+ work item types and therefore will have 2+ workflows. With the General service management template, you really only get 1 (technically 2, but I’m trying to keep it simple).

The blank template has the most basic of request types, the email so that means you’ll have to build everything else from the ground up.

Long story short, if your team needs a true ITSM solution, you’ll want to use the ITSM template and make sure you are on the Premium/Enterprise version of JSM. If you don’t have Premium, but still want an ITSM solution, then Basic Service Management will be best. If you don’t care for ITSM, but also don’t want a completely blank slate, then the General service management is the right template.

Finally, if you have a specific use case for JSM, then the Blank or any one of the use case specific templates might be right for you.

Setting Up a Team-Managed JSM Space

Step 1: Creating a Team-Managed JSM Space

- Click on the + icon next to spaces.
- Select Service Management on the left
- Pick your favorite Service Management template
  - Refer to the guide that breaks down all the different templates
- Click on Use Template, once you have selected the template you are going to use
- Select Team-managed
- Provide the Name, Key, Team Type, and Access/Permissions
  - The Team Type should be whatever is closest to the team that is going use this space

Step 2: Creating a New Request Type

- Go to Space Settings → Request Management → Request types

Disclaimer: You'll need to be a space administrator in order to create a request type.

- Click on + Add request type.
- You can either create from template or create blank. I normally always create from blank, but just like with picking a JSM space template, choose whatever makes more sense to you.
- Provide a Name, Description, and Icon for your new request type.
  - The name of your new request type should be some type of help that your end user is going to need from your team. For example, if you are running an IT help desk, you may make a request to enable end users to submit for a new computer, or maybe a new license request.
- Click on Add when you are finished.

Step 3: Setting up a Request Type

- Go to Space Settings → Request Management → Request types
- Select an existing request type, or if you just created a new request type, you'll already be in the configuration for that request type.
- For each request type, you want to modify/configure the fields that the end users will eventually fill out. Some of these fields will be important, required fields, others will be optional.
- You can use any existing fields or you can create a new field. Place them in the Customer Form section. Optionally, you can build a form and create/place your fields there.
- For every field (new or existing) that you add to your request type, you can choose to:
  - Display a different name
  - Provide "helper" text
  - Make the field required
- Once you finish configuring your request type, save your change and preview the form by clicking on View.
- Repeat for any other request types you want to configure.
- Refer to setting up a team-managed space for workflow configurations.

Setting Up a JSM Space for ITSM Run Book

Space Setup

When creating a company-managed JSM space, select the IT Service Management (ITSM) template. This gives you preconfigured elements aligned with ITIL practices.

The ITSM template organizes requests into four main work categories:

- Service requests (e.g., new laptop, access to systems)
- Incidents (e.g., outage, performance degradation)
- Problems (e.g., recurring email failures)
- Changes (e.g., patching, deployments, upgrades)

Core Work Types

The ITSM template comes with these standard work types (recommended to keep them as is):

- Service Request – User requests for help, information, or service (new software, access).
- Incident – Something broken or degraded that needs fixing quickly.
- Problem – Root cause investigation of recurring issues.
- Change – Planned adjustments to infrastructure, applications, or processes.
- Task – General work item not tied to ITSM practices.

You can add additional work types as appropriate for your specific processes.

The work types are different than the request types from the first step.

Recommended Workflows

Each Work type has its own workflow. Here's a suggested setup:

Service Requests

Statuses: To Do → In Progress → Waiting for Customer → Closed

Notes: Keep simple for speed; use automation to auto-resolve if customer doesn't reply.

Incidents

Statuses: To Do → In Progress → Waiting for Vendor → Resolved → Closed

Notes: Consider adding Major Incident path with escalations and comms.

Problems

Statuses: To Do → Under Investigation → Known Error → Resolved → Closed

Notes: Use problem linking to incidents.

Changes

Statuses: To Do → Assess → Approved → In Progress → Completed → Closed

Notes: Add CAB Review status if you run formal CABs.

Tasks

Simple To Do → In Progress → Closed.

Recommended Fields

Here are must-have fields by work type:

- **Service Request:** Request Type, Priority, Description, Assets (optional).
- **Incident:** Summary, Priority, Description, Affected Services, Linked Problems.
- **Problem:** Summary, Description, Priority, Root Cause, Workaround.
- **Change**: Summary, Priority, Risk, Impact, Implementation Plan, Backout Plan, Start/End Date, Approvers.
- **Task:** Summary, Description, Assignee, Due Date.

Use Forms in JSM to make these fields easier for customers to fill in.

Recommended Request Types

These are common request types you should configure in your ITSM portal:

Service Requests

- Request new hardware (laptop, monitor, phone)
- Request new software / access to system
- Password reset
- General question
- Incidents
- Report an outage
- Application not working
- Performance issues

Problems

- Report recurring issue
- Request root cause analysis

Changes

- Standard change (pre-approved, low risk)
- Normal change (needs approval)
- Emergency change (fast-tracked, incident-related)

Tasks

- General IT work item

Automations to Enable

- Auto-assign work items based on request type (e.g., network → network team).
- SLA breach alerts.
- Auto-close service requests after X days of inactivity.
- Post-resolution surveys. (Survey Monkey)
- Link major incidents to problems automatically.

SLAs to Configure

Incidents: Time to first response (30 min), Time to resolution (4 hrs for high priority).
Service Requests: First response (8 hrs), Resolution (3 business days).
Problems: Time to acknowledge (24 hrs).
Changes: Time to review/approve (2 business days).

JSM Space Setup Glossary

Space	A container for your service desk work. In JSM, a space holds all requests, queues, SLAs, automations, and portal settings for one team or service.
Team-Managed space	A self-contained JSM space where configuration (workflows, fields, request types) is managed at the space level. Perfect for small/independent teams.
Space Settings	The admin panel for the space. This is where you configure request types, SLAs, queues, automations, notifications, and portal branding.
Request Type	The customer-facing form customers use in the portal. Each request type maps to a Jira work item type (e.g., "Need a laptop" = Service Request). You customize request types to control what customers see.
Work Item Type	The internal classification of work. JSM comes with common work types: (Service Request, Incident, Problem, Change) You can add custom ones depending on your team's needs
Portal (Customer Portal)	The website where customers submit and track requests. You configure it with request types, instructions, and branding.
Help Center	If you have multiple JSM spaces, the Help Center shows customers all the portals in one place.
Queues	Custom views for agents. Queues are built from filters and help agents organize work (e.g., "Unassigned tickets," "High priority," "Waiting for support").
SLA (Service Level Agreement)	Time-based goals you define per space. SLAs are measured automatically in JSM (e.g., "Respond within 2 hours," "Resolve within 5 days").
Approvals	Workflow steps where requests pause until certain people approve/reject. Common in **Change Management** projects.
Channels	Ways customers can create requests: • Portal (default) • Email (set up during space configuration) • Chat/Slack/Teams (if integrated) • API (for system-to-system requests)

JSM Spaces Creating Mistakes To Avoid

1. Confusing Request Types with Work Item Types

- **Mistake:** Admins create dozens of work item types thinking they'll show up on the portal, or they re-use software space work item types without tailoring them.
- **Why it's a problem:** Customers see a messy, technical portal ("Incident," "Bug," "Change") instead of clear language ("Need new laptop," "Report outage").
- **Fix:** Keep a small set of work item types (Service Request, Incident, Problem, Change) and build customer-friendly request types on top.

2. Overloading the Portal with Too Many Request Types

- **Mistake:** Making 30+ request types for every small variation ("Request mouse," "Request keyboard," "Request monitor").
- **Why it's a problem:** Customers get overwhelmed and don't know what to pick → they abandon the portal and email your team instead.
- **Fix:** Start broad ("Request new hardware") and use forms or fields to capture specifics.

3. Not Setting SLAs Properly

- **Mistake:** Leaving default SLAs in place or setting unrealistic ones (like "Resolve all requests in 1 day").
- **Why it's a problem:** SLAs drive team accountability and reporting. If they don't match business expectations, you'll either always fail or not measure the right thing.
- **Fix:** Define SLAs with stakeholders (e.g., "Critical incidents: 1 hr response, 4 hr resolution").

4. Forgetting to Configure Business Hours

- **Mistake:** Not setting calendars → SLAs tick 24/7.
- **Why it's a problem:** Tickets logged at 10pm show as "breached" the next morning.
- **Fix:** Configure working calendars to match team hours before SLAs go live.

5. Overcomplicating Workflows

- **Mistake:** Adding too many statuses (e.g., "Pending Triage," "Pending Assignment," "Pending Work," "Pending QA," "Pending Closure").
- **Why it's a problem:** Agents ignore them, and customers get confused.
- **Fix:** Keep workflows lean → usually "Open → In Progress → Waiting for Customer → Resolved."

6. Not Customizing Customer Notifications

- **Mistake:** Leaving default Jira emails on.
- **Why it's a problem:** Customers get spammed with cryptic Jira emails ("AP-42 status changed to In Progress"). They stop engaging.
- **Fix:** Customize notifications with plain language ("We're working on your request. We'll update you soon.").

7. Mismanaging Permissions

- **Mistake:** Letting anyone browse the space or forgetting to restrict agent licenses.
- **Why it's a problem:** Costs skyrocket when too many users become "agents" by accident.
- **Fix:** Assign roles carefully → only true support staff should be agents. Everyone else = customers.

JSM Request Types Guide

IT Service Desk / IT Support

- Report an Issue – "Something's broken."
- Request New Hardware – laptops, monitors, peripherals.
- Request New Software – apps, licenses, installations.
- Password Reset – unlock or reset accounts.
- Access Request – system, application, or shared drive access.
- Email/Account Setup – create or remove user accounts.
- Network/Connectivity Issue – Wi-Fi, VPN, or internet issues.
- Printer/Device Issue – printing, scanning, or peripherals not working.

HR Service Desk

- Onboarding Request – new hire setup (accounts, equipment, permissions).
- Offboarding Request – revoke access, collect equipment.
- Payroll/Compensation Inquiry – salary, benefits, deductions.
- Leave/Time Off Request – PTO, sick leave, vacation.
- Policy Question – HR policies or employee handbook queries.
- Employee Relations Concern – workplace issues or conflict reporting.

Finance / Procurement Service Desk

- Purchase Request – order equipment, software, or services.
- Expense Reimbursement – submit expenses for approval.
- Invoice Question – billing or payment inquiries.
- Budget/Forecast Request – ask for financial reports or allocations.
- Vendor Management Request – add or update vendor details.

Facilities / Operations Service Desk

- Maintenance Request – broken furniture, HVAC, lighting.
- Office Move Request – moving desks or departments.
- Badge/Access Card Request – building or room access.
- Event Setup Request – conference rooms, catering, equipment.
- Health & Safety Issue – hazards or safety concerns.

Marketing / Creative Service Desk

- Design Request – graphics, slide decks, branding assets.
- Campaign Request – launch or support for marketing campaigns.
- Content Request – blogs, social posts, copywriting.
- Website Update – content changes, bug fixes.
- Event/Trade Show Request – collateral, logistics, registrations.

General Service Desk / Enterprise Support

- General Inquiry – catch-all request type.
- Request Information – policies, documentation, FAQs.
- Submit Feedback – suggestions or complaints.
- Report a Problem – any non-IT issue.
- Service Complaint – escalate a poor experience.

JSM Workflows With Approvals Guide

JSM allows JSM admins to create workflows with approvers. These types of workflows are technically available in team-managed Jira spaces, but they work much better in JSM. You select a status that you need an approver on and then you select who will be the approver(s). Then, if they approve, you determine which status the work item will transition to. If they reject, you'll also need to determine which status the work item should transition to. The nice thing about approval workflows in JSM is that anyone can be an approver, as long as they have an agent or customer license. An automated email is sent out to the approver and they can approve through the email, through the JSM portal, or if they are an agent, on the work item directly.

Pro tips: If you are creating a new workflow, or are using a workflow that hasn't yet been associated with a JSM space, you will not be able to utilize the approval feature. Make sure you create all your statuses and transitions first and then apply the workflow to a JSM space. Once you **publish**, you'll be able to go back into the workflow and apply the approval feature.

For your approver, you can use the out of the box Approvers field, or, if you want a group of approvers, you can use the approvers group. Additionally, you can use any custom User Picker field.

Determine how many total users need to approve. JSM will allow you to select a first come to approve situation, or you can enforce that everyone approves. If you are using group approvals, you can specify if only 1 person from each group needs to approve, or if all users need to approve.

Depending on your requirements, you may want to exclude the reporter or assignee from being able to approve so that they don't "accidentally" circumvent your approval process.

JSM SLAs Guides

What Are SLAs in JSM?

- An SLA (Service Level Agreement) is a time-based goal you set for requests in your JSM space.
- They help track response times and resolution times to meet customer expectations.
- SLAs in JSM are automated, visible in work items, and can trigger notifications or escalations.

Key SLA Concepts

- Start condition: When the SLA timer begins (e.g., work item created, status = “Waiting for support”).
- Stop condition: When the SLA timer stops (e.g., work item resolved or closed).
- Pause condition: When the SLA timer is temporarily paused (e.g., status = “Waiting for customer”).
- Calendars: Define working hours and holidays so SLA timers reflect your team’s schedule.
- Goals: Different time targets based on priority, request type, or other conditions.

Why Use SLAs?

- Set clear expectations for customers.
- Prioritize work by urgency and importance.
- Increase accountability — everyone sees SLA progress directly in the ticket.
- Automate escalations to managers when tickets breach SLAs.
- Improve reporting — track SLA success rate over time.

Common SLA Examples

- Time to First Response
  - Start: Work item created
  - Stop: First public comment added by agent
  - Goal: High = 1 hour, Medium = 4 hours, Low = 1 business day
- Time to Resolution
  - Start: Work item created
  - Stop: Work item resolved/closed
  - Goal: High = 4 hours, Medium = 3 business days, Low = 5 business days
- Time in Status (custom)
  - Start: Work item enters “In Progress”
  - Stop: Work item moves to “Waiting for Customer”
  - Goal: Track time spent actively working

Best Practices for SLAs

- Start small – focus on 2–3 critical SLAs (First Response + Resolution) before adding more.
- Use pause conditions wisely – prevents unfair breaches when waiting on the customer.
- Customize by request type – not all requests need the same SLAs.
- Set realistic goals – align with team capacity and customer expectations.
- Monitor breaches – use automation to escalate tickets close to breach.
- Train agents – make sure they understand SLA timers shown in work items.

How to Configure SLA's

Step 1: Open SLA Settings
Go to your JSM space → space settings → SLAs.

Step 2: Create a New SLA
Click Add SLA.
Name your SLA (e.g., "Time to First Response").

Step 3: Define SLA Conditions
Start counting time when: Select trigger (work item created, status changed, etc.).
Stop counting time when: Select resolution conditions (resolved, closed, comment added).
Pause counting time when: Optional (e.g., "Waiting for customer").

Step 4: Apply Goals
Add multiple goals using JQL conditions.
Example:
Priority = High → 1h
Priority = Medium → 4h
Priority = Low → 1d

Step 5: Attach a Calendar
Create a Calendar with your team's working hours (e.g., 9 AM – 5 PM, Mon–Fri).
Add holidays if needed.
Apply this calendar to SLA so timers don't count downtime.

Step 6: Save & Test
Apply SLA → Save.
Create test work items with different priorities to verify SLA timers work as expected.
Depending your start SLA setting, you may not immediately see an SLA, it will only show up once the start SLA has been triggered.

JSM Licensing Guides

License	Definition	What they can do
Agent	Your support team — the people working on requests.	• View and work in queues. • Respond to customer requests (internal & external comments). • Be assigned work items. • Set SLAs, run reports, configure space settings. • Approve requests.
Customers	End users who submit requests through the portal, email, or chat.	• Submit requests (via portal, email, chat, or integrations). • Track their own requests in the portal. • Comment/reply on their requests. • See requests raised by their organization (if configured).
Stakeholders	Internal stakeholders (like executives, managers, or business leaders) who only need visibility into requests and reports.	• Can view requests and the customer portal. • Can track progress of work items they're interested in. • Can receive notifications. • Cannot work requests (no transitions, assignments, or internal comments).
Collaborators	Internal team members (like developers, HR, finance, or other business staff) who occasionally need to help resolve JSM requests.	• Can be @mentioned in comments. • Can add internal comments to help agents. • Can browse and view requests. • Cannot manage queues, SLA rules, or customer communications.

JSM Customers and Organizations Guide

Customers

- **Who they are:** The people submitting requests into your service space.
- **License cost:** Free (customers do not need a Jira license).
- **What they can do:**
  - Log in to the customer portal or send emails to create requests.
  - View and track their own requests.
  - Comment on their requests (publicly with agents).
  - If allowed by settings, view requests raised by others in their organization.
- **Best use case:** Employees inside your company (for IT/HR requests) or external clients (for support).

Organizations

- **What they are:** A grouping of customers.
- **Purpose:** Makes it easier to manage who sees what.
- **What they can do:**
  - All customers in an organization can (if allowed by space settings):
  - Raise requests on behalf of the org.
  - View all requests submitted by others in their org.
  - Great for companies that want visibility across all their requests (e.g., “Acme Corp” wants all employees to see and track IT issues).
- **Best use case:** Group customers by company, department, or client account.

Example

Let’s say you run an IT Helpdesk:

- **Customer:** John Smith (employee at Acme Corp) submits a ticket: “Laptop won’t start.”
- **Organization:** John is part of the “Acme Corp” organization. By default, Acme employees in that org can see each other’s requests and avoid duplicates.
- Or, if you run an external support desk:
- **Customer:** Jane Doe (client from Widget Inc) submits: “Need help setting up integration.”
- **Organization:** Jane belongs to “Widget Inc.” Everyone at Widget Inc can see all support requests their colleagues opened.

JSM Assets Guide

What is JSM Assets?

- Assets is Jira's CMDB (Configuration Management Database) and asset management tool.
- It lets you track anything your team supports or manages: laptops, software licenses, office equipment, servers, vendors, contracts, even employees.
- Assets link directly to requests so agents always know who owns what and what's impacted.

Why Use Assets?

- Better context for requests – e.g., when a user requests IT help, agents can instantly see their assigned laptop, software, and warranty info.
- Reduce resolution time – no more chasing down asset details; it's tied to the ticket.
- Track dependencies & impact – see relationships between systems (e.g., if Server A goes down, it impacts Application B and Team C).
- Central source of truth – move away from scattered spreadsheets.
- Improve reporting – understand cost, usage, and lifecycle of assets.

Asset Terms to Know

Term	Meaning	Example
Object	An individual item tracked in Assets.	Laptop "Dell-12345"
Object Type	Category/group of similar objects.	"Laptops," "Employees," "Vendors"
Schema	A container for objects & their relationships. Think of it like a database.	"IT Assets Schema"
Attributes	Data fields describing objects.	Serial number, purchase date, location
Relationships	Links between objects.	Employee → owns → Laptop
Asset field	A field in JSM requests that lets customers or agents select an object from Assets.	"Which device needs repair?"

Getting Started with Assets

Step 1: Enable Assets
In JSM Premium or Enterprise, Assets is included.
On the left navigation, click on Assets. Assets is now it's own App.

Step 2: Create a Schema
Fine Schemas at the top left corner and click on the + button.
Use a template or create a blank schema.
Name it based on scope (e.g., "IT Assets," "HR Resources," "Facilities Equipment").
Provide a key, or accept the default that Jira creates for you.
Optionally add a description.
Click on the Create Schema blue button.

Step 3: Define Object Types
Within your schema, create object types (categories).
Example for IT:
Hardware → Laptops, Desktops, Phones
Software → Licenses, Applications
People → Employees, Vendors
Under Schema Tree, click on + Add object type
Provide a Name, an Icon. Optionally, provide a Parent and Description.
Add attributes to describe each object type (e.g., for laptops: model, serial number, purchase date, owner).

Step 4: Import or Add Objects
Manual entry → Good for small setups.
- Click on the blue Create Button.
- Select the Schema, the Object Type, an icon, and the name for the object.

CSV import → Upload large lists of assets.
Integrations → Use marketplace apps or APIs to sync with tools like Intune, SCCM, Jamf, or AWS.

Step 5: Create Relationships
Link objects together:
Employee → owns → Laptop
Application → runs on → Server
In the attributes for an object, add an attribute and select object as the format.
Select one of the default references or create your own reference to associate two objects.
Relationships let you visualize dependencies and troubleshoot faster.

Step 6: Connect Assets to JSM Requests
Go to Request types in your JSM space.
Add an Assets custom field (e.g., "Select your device").
Configure it to pull from the right schema/object type.
Example: On "Hardware issue" request type, show a dropdown of laptops assigned to that customer.

JSM Forms Conditional Fields and Sections

JSM forms help you structure requests so customers provide all the information your team needs. Using forms reduces back-and-forth, ensures consistent data, and improves resolution times. Forms don't exactly replace the need to add fields to request types, but they make it a lot easier. Forms in JSM are also much more powerful than forms in Jira. You can create local fields to the form only, or you can create custom fields and reference those fields in the form.

1: Creating a Form

- Go to your JSM space → space settings → Request management → Forms.
- Create a new from from blank or from a template.
- I'm going to go with a blank to begin.
- Feel free to add any fields to build out your form.
  - Text fields (short answer, long answer)
  - Dropdowns (e.g., hardware type: Laptop, Monitor, Mouse)
  - Checkboxes / Multi-selects (e.g., software access needs)
  - Date picker (for scheduling requests)
  - Mark fields as required if they are essential for fulfilling the request.
- The Display name is the name of your "new" field. Replace Label with whatever name you want.
  - If you want to link the "new" field with Jira, then the field needs to exist before you use it in the form. If you don't already have the field available in Jira, you'll want to go and create the field first (will need a Jira admin for this) and then you can use this.
- Forms are great because it allows non Jira admins to create fields however, the fields that are in the form only aren't searchable by JQL. So, if you want to query the data the end users submits or if you want to include any of that data in a dashboard, then a Jira admin needs to create the field in Jira first.

2: Conditional fields and sections

- If you are using a select list (dropdown) field, you can dynamically make sections appear on your form based on the value the user selects. This is one the best JSM Forms feature that is missing from Jira.
- Click on Add section.
  - Provide a name for the section and switch the Show section to Conditionally.
  - Select the field that you want to drive whether the section is shown or not.
  - Select which value(s) will trigger the section to be displayed.
  - Click on any white space and the section will now be created.
  - You can now add fields that will only be displayed when the previous field matches your show section criteria.
  - Once you finish adding those fields, you have the option to add another section that is either conditionally shown (based on the new fields you just added. Alternatively, you can add a section that is always shown. Finally, you can simply just end your form there and be done.
- When finished, click on the Save changes button.
- Preview the form by clicking on Preview at the top.
- Test and confirm that the section that are supposed to be conditional work based on your logic.

JSM Product Administrative Setting

Configuration	General JSM settings, but most importantly, determines if emails become new requests or if they contain the work item key, get appended as a comment
Email Requests	Set up settings for how emails are handled.
Customer Access	Configure how new accounts are created for customers to send requests and access portals. These settings impact all service spaces on this site
Authentication	Configure customer login settings to portals. These settings impact all Jira Service Management spaces on this site
Organizations	Configure global organizations to organize Customers
Incident Management	Set up tools and view incident management settings for all your IT service management spaces.
Knowledge Base Permissions	Configure how space admins can manage who can view and edit articles in their knowledge base spaces.
Compliance Settings	Choose to hide sensitive and confidential information. Also, configure HIPAA notification alert settings.
Feature Usage	Track how many assets and virtual agents your team is using.

Knowledge Base Setup Guide

Benefits of Using a Knowledge Base in JSM

- Customers often find answers themselves without creating a request.
- This reduces repetitive “How do I reset my password?” type tickets.
- Agents can search KB articles directly inside Jira and share them with customers in one click.
- Cuts down response time and keeps answers consistent.
- Improved Customer Experience
- Customers get immediate, 24/7 access to solutions.
- No need to wait for an agent.
- As agents spot repetitive issues, they can create KB articles to prevent future tickets.
- Over time, this builds a rich self-service library.

How to Configure a Knowledge Base in JSM
Requirement:

- You’ll need Confluence (Atlassian’s wiki product). JSM integrates directly with it. If you don’t have it, you’ll need to add Confluence to your Atlassian site.
- You’ll need to open/loosen your Confluence permissions so that Customers can access information

Step 1: Link Confluence to Your JSM space

- Go to your JSM space → space settings → Channels and Self Service
- Select Knowledge base from the menu → Try Now → Confluence
- Choose Link a space or Create a new space.
- You can also link to a Confluence on a different site (but this is more risky as your users might not have access to that site).
- Feel free to add more spaces as needed.

Step 2: Configure Permissions

- Who can view articles?
  - Confluence users - must have a Confluence license.
  - Internal only - Any logged in user with a JSM license
  - Anyone - Customers (public articles visible in the portal).
- Who can create/edit articles?
  - Usually, agents and KB managers.

Step 3: Add KB Articles

- In the JSM space, go to Knowledge Base
- Select/group articles so that end-users can access the articles easily.
- As long as Show Suggestions is enabled, when a user starts to submit a request, a “related” Confluence article will show up.

Pro tip: Start with the top 10 most common requests (like password reset, access request, onboarding) — document those first. This gives you maximum impact right away.

revyz

COMMAND CENTER
SECURITY COMPLIANCE PLATFORM

https://www.revyz.io

GETTING STARTED WITH JIRA PRODUCT DISCOVERY ADMINISTRATION

Setting Up a JPD Space Check List

1. Create the Space

- Hover over Spaces on the left nav → Click on the + button → Select Product management
- Choose Product discovery template → Use template
- Name the space and define the space key
- Set the desired access level for the space (same as team-managed spaces).

2. Define Fields & Custom Attributes

- Review default fields: Impact, Effort, Confidence, Goal, Target date
- Add custom fields as needed, e.g.:
  - Customer Value
  - Strategic Fit
  - Revenue Potential
  - Risk Reduction
- Standardize scales (e.g., 1–5, t-shirt sizing, percentages)

3. Set Up Views

- Ideas backlog (default)
- Roadmap (Kanban/Timeline for visibility)
- Impact vs Effort matrix (2x2 view for prioritization)
- Scoring views (weighted score formula across fields)
- Executive summary (filtered, simplified view for leadership)

4. Configure Workflows / Statuses

- Default statuses: Draft → Under Consideration → Planned → In Progress → Done
- Adjust to match product team process (e.g., Parked, Validated, Not pursuing)
- Ensure statuses align with reporting and roadmap needs

5. Set Up Prioritization Framework

- Agree on scoring model (RICE, WSJF, custom)
- Configure formula fields if needed
- Create saved views for ranking ideas

6. Set Permissions & Collaboration Rules

- Define who can:
  - Create ideas (product team vs everyone)
  - Comment/vote/react (often open wider to stakeholders)
  - Edit fields (restricted to product managers)
- Add stakeholders and contributors to space
- Encourage voting & insights collection

JPD Space Setup Glossary

Core Concepts

- **Idea** – The main work item in JPD. Represents a potential product improvement, feature, or initiative.
- **Insight** – Supporting evidence (customer feedback, market research, support tickets, analytics) attached to ideas.
- **Views** – Saved configurations of how ideas are displayed (table, board, timeline, matrix). Used for prioritization, roadmaps, or stakeholder presentations.
- **Roadmap** – A visualization of ideas over time, often grouped by quarter, release, or initiative.

Fields & Prioritization

- **Impact** – How valuable or beneficial an idea is if implemented.
- **Effort** – Estimated work required to deliver the idea (can be story points, t-shirt size, hours, etc.).
- **Confidence** – How sure the team is about the accuracy of impact/effort assumptions.
- **Scoring Model** – A prioritization framework applied to ideas (e.g., RICE: Reach, Impact, Confidence, Effort or WSJF: Weighted Shortest Job First).
- **Formula Field** – A calculated field in JPD that applies a mathematical formula to multiple attributes (e.g., Impact ÷ Effort).

Collaboration & Permissions

- **Contributors** – Team members who can create and edit ideas.
- **Stakeholders** – Users who can comment, vote, or view ideas but not manage fields or workflows.
- **Voters/Reactions** – Lightweight stakeholder input to signal interest or priority.
- S**pace Lead** – Primary owner of the JPD space.

Advanced JPD Features

- **Matrix View** – 2x2 chart (e.g., Impact vs Effort) for visual prioritization.
- **Timeline View** – Calendar-style view for roadmap planning.
- **List/Table View** – Spreadsheet-style format for scoring and sorting.
- **Grouping** – Organizing ideas by category, owner, status, or strategic goal.
- **Filtering** – Narrowing views to only show relevant ideas (e.g., high impact, Q1 delivery).
- **Delivery Link** – A connection between a JPD idea and Jira Software work item(s), usually an Epic.

Setting Up a JPD Space for Product Management Run Book

Core Prioritization Fields

- Impact (1–5 scale) – potential business/customer value
- Effort (1–5 scale) – rough dev complexity
- Confidence (High/Med/Low) – certainty in assumptions/data
- Reach (# of customers affected) – optional if you want RICE
- Score – auto-calculated (e.g., RICE or ICE formula)

Workflow/Status Fields

- Status – (Idea → Prioritized → In Discovery → Ready for Delivery → In Delivery → Shipped)
- Category – Bug / Feature / Improvement / Experiment / Other
- Theme/Initiative – to group ideas under product pillars

Metadata Fields

- Owner (PM) – person responsible for the idea
- Target Audience / Persona – optional if you're customer-focused
- Source – where the idea came from (Customer, Sales, Internal, Market Trend, etc.)
- Insights (linked) – feedback, research, or tickets connected to the idea
- Delivery Link – link to Epic(s) in Jira Software

Views You Should Create in JPD

Idea Intake View

- Layout: List
- Fields: Idea, Owner, Category, Source, Created Date
- Purpose: Triage new submissions, clean duplicates, and tag appropriately.

Prioritization View (RICE/ICE Matrix)

- Layout: Table
- Fields: Impact, Effort, Confidence, Reach, Score
- Purpose: Compare and prioritize ideas based on scoring framework.
- Tip: Sort by Score or add conditional formatting for clarity.

Strategic Alignment View

- Layout: Board or Table
- Fields: Theme/Initiative, Status, Impact, Effort
- Purpose: Show how ideas ladder up to product strategy pillars.

Delivery Tracking View

- Layout: Timeline or Board
- Fields: Status, Delivery Link, Owner
- Purpose: Connect ideas to Jira Software epics/stories and track progress from discovery to shipped.

JPD Spaces Creating Mistakes To Avoid

Strategic Mistakes

- Treating JPD like a backlog dump
  - Teams often import every idea, feature, or request without prioritization → JPD becomes overwhelming and useless for decision-making.
  - Fix: Define what qualifies as an "idea" and set entry criteria.
- Not defining the product vision upfront
  - Without a vision/strategy, prioritization fields like "impact" or "value" become arbitrary.
  - Fix: Align fields and views to product goals before adding ideas.
- Mixing customer feedback with ideas
  - Some teams create duplicate or scattered feedback items instead of linking feedback directly to ideas.
  - Fix: Use the Insights feature to connect customer input to ideas. Use a different JPD space for customer feedback to keep things organized.

Configuration Mistakes

- Overcomplicating fields
  - Adding too many custom fields early → every idea becomes heavy to create/manage.
  - Fix: Start simple with core fields (Impact, Effort, Confidence, etc.) and expand later.
- Not tailoring views for stakeholders
  - PMs, execs, and engineers all want different views. If everyone sees the same messy list, it creates confusion.
  - Fix: Build separate views (Roadmap, Prioritization matrix, Delivery board, etc.) per audience.
- Failing to connect JPD to Jira Software
  - Ideas stay in JPD but aren't linked to Epics/Stories in delivery → no visibility of progress.
  - Fix: Set up delivery link rules or automation to sync with Jira Software spaces.

Collaboration Mistakes

- Not giving the right permissions
  - Either too strict (only PMs can add ideas) or too open (everyone spams ideas).
  - Fix: Decide roles (contributors vs voters) and control who can add vs vote/comment.
- Ignoring voting and insights
  - Teams use JPD just as a "feature wishlist" and forget to capture actual stakeholder/customer voices.
  - Fix: Actively encourage voting and connect customer feedback tools (e.g., Slack, Intercom, Salesforce).
- Not closing the loop with stakeholders
  - Ideas go in, but stakeholders never see updates → they stop contributing.
  - Fix: Use statuses and share roadmap views so people see progress.

Reporting Mistakes

- Not defining prioritization criteria
  - Relying on gut feeling instead of using fields like Reach, Impact, Confidence, Effort.
  - Fix: Agree on scoring models (RICE, ICE, custom) early.
- Treating JPD as static
  - Teams create a roadmap once and never update → it loses trust.
  - Fix: Review and groom ideas regularly (weekly/biweekly) to keep the project alive.
- Forgetting executive-friendly communication
  - Leaders want clear roadmaps and high-level priorities, not the raw idea backlog.
  - Fix: Build "Executive views" with grouped/rolled-up ideas.

JPD Product Administrative Setting

Setting	Definition
Access	Jira admins can control who is able to see views in JPD. Admins can configure a “contributor” license that is free and enables others in your organization to view JPD views. There is a second setting that is open to all users on the internet. More caution should be exercised when enabling this.
Global Fields	Global fields, not quite the same as “custom” fields in company-managed fields, allow you to create fields that can be used across JPD spaces. This is slightly better than team specific fields, but not as good as global fields from company-managed Jira spaces.
Dates	Allows a Jira admin to set the start date for the year. This is a global setting so it would apply to all JPD users. Once set, this will then guide the quarterly roadmap.
Beta Features	Whenever Atlassian releases new features for JPD, your Jira admin can come and turn those features on. Given Atlassian’s current trends, expect most of these features to be for Premium subscribers. Also, keep in mind these features are in beta, I wouldn’t depend on them for production systems as they may change over time and/or break things.

GETTING STARTED WITH CONFLUENCE ADMINISTRATION

S R E T S N O M S N O I S S I M R E P E G A P
N V H E M O G N I T T E S E C A P S M F N R R
O E S S E C C A S U O M Y N O N A R O P D O E
I P T O O L S P U R G N H I M T G G S E I I N
S A S O I T O A V U C H A W T E L O D A A D T
S R G A M T G T O I O I B E R T A L P B T T P
I E N C O E M O T R A S E O B P S I U O I I A
M T P R T O M P C R H P O E L L U L C M C L R
R P T A I A C L T I A E B S E I O D A A A G E
E E S R G A E S I R O G E T A C E C A P S S C
P G E O S T L U S C C U R O A I R S E T N E P
L A N C A T E S D E E A E L E R T E G A P O T
A P T C M C A A S G P L C E O G N C L P T C S
B D I A R A I N E I E C S S O O P H O S E I R
O T E R C T O O E H I A A C N A R A R M S R E
L L N L O T E L T H S T E I C C A E C D O E G
G O A T L P O T E G S R P T A R H C E C C C A
H B A C M I O I A M N L E L T G A S S I S T P
N I E E A D O A R I O A L E I P L A B E L S S
A H C S P A C E A R C H I V E O D S S A T S S

Space
Child Page
Parent Page
Macros
Live Doc
Database
Space Permissions
Anonymous Access
Page Restrictions
Global Permissions
Page Tree
Blueprints
Page Templates
Labels
Space Categories
Space Settings
Star space
Automation
Space Directory
Content Tools
Space Archive
Whiteboard
Homepage
Page
Space Trash

Determining Confluence Template Guide

When creating a Confluence Space, there are 3 different types of “templates” that you can use. Picking one of these templates will govern which pages are automatically available under your space content page tree when your space is created. It doesn’t really matter which template you pick as all the pages that are automatically created can be renamed, removed, or you can just create the ones your team will actually use. These are just there for inspriation.

Collaboration

The collaboration template provides you with the following pages:

- Template - Project Plan - An outline for a project plan that can be used for other project plans. You make a copy of this page and nest future project plans under this page.
- Template - Decision Document - An outline for how your team can handle major decisions.
- Template - Meeting Notes - An outline for how to take meetings.

These templates can be modified after the space is created. Make sure you make any modifications before team members start to use them.
Optionally, you can remove these and create blueprints (templates) that your team can utilize, more on this later in this book!

The Knowledge base is more technical and personal as opposed to the more open and collaborative nature of the collaboration template.

- Template - How-to guide - A super basic template that contains an instructions section to help you document a step by step guide.
- Template - Troubleshooting - Very similar to the how-to guide, but focused on providing a template that you can use to document troubleshooting steps.

The Custom template is the template you want to use if you don’t want any preconfigured templates. The custom template is just an empty space that you and your team get to configure however you want.

I usually just use the Custom template whenever I create a new Space.

What to ask stakeholders before building a Confluence space?

A Confluence space is much more flexible, you more carefully want to structure the pages of your space. How you structure pages will determine how well organized your team documentation is.

Think about the “big buckets” of content you’ll have:

- Reference Information – policies, SOPs, how-tos
- Collaboration Work – meeting notes, project docs, brainstorming pages
- Announcements/Updates – team updates, release notes
- Knowledge Capture – FAQs, troubleshooting, lessons learned
- Keep the homepage as a navigation hub, not a wall of text.
- Page tree should have 3–4 top-level categories max (avoid too much nesting).
- Use templates for consistency (meeting notes, space briefs, decision logs).
- Consider how people will search vs browse → design for both.

Questions to ask

- Do you want the space open to everyone, or restricted by team?
- Should outsiders (e.g. other departments, customers) have view-only access?
- Will some content need page-level restrictions (sensitive docs, drafts)?
- Will guests be able to access this space?
- Are you going to share individual pages with Public Links?
- If you’ll have multiple spaces (HR, IT, Product, Marketing), agree on naming conventions and layout patterns so people don’t feel lost.
    - Example: All spaces start with “About This Space,” “How to Use This Space,” and “Index Page.”
- Who owns the space? (space admin / content steward)
- Will pages be archived, reviewed, or pruned on a schedule?
- If this is documentation → consider versioning or labeling for “current vs outdated.”
- Plan how to use labels/tags to improve searchability.
- If you’re scaling knowledge management → consider using Confluence apps like Metadata or Scroll Documents for structured info.
- If this space supports a Jira space → how will you link work items (e.g., release notes, requirements, retrospective docs)?
- Will you embed Jira boards, roadmaps, or reports in Confluence?

Setting Up a Confluence Space Check List

Purpose & Audience

- What is the primary purpose of this space? (Documentation, Team hub, Knowledge Base, Project tracking, etc.)
- Who is the main audience? (Internal team, whole company, leadership, external customers)
- Will the content be long-term reference or short-term collaboration?

Content & Structure

- What are the big content categories I'll need? (e.g., Policies, Meeting Notes, Roadmaps, FAQs)
- Do I have a homepage plan? (Navigation, quick links, intro text)
- How will the page tree be structured? (Max 3–4 top-level categories, shallow hierarchy)
- Do I need to set up templates for common content? (Meeting notes, decision logs, specs, etc.)

Permissions & Access

- Who should have view access?
- Who should have edit/create access?
- Do I need any restricted pages (sensitive or draft content)?
- Who will be the space admin/owner responsible for upkeep?

Findability & Consistency

- Will I use labels/tags for search optimization?
- Do I need a consistent naming convention for pages?
- Is this space aligned with other spaces (look, feel, naming)?

Maintenance & Lifecycle

- Who is responsible for reviewing and archiving old content?
- Do I need a content lifecycle policy (e.g., quarterly reviews)?
- How will people know which pages are current vs outdated?

Integration & Extras

- Do I need to link this space with Jira spaces/work items?
- Will I embed Jira boards, roadmaps, or reports?
- Do I need any apps/macros (Calendars, Draw.io, Metadata, etc.)?

Confluence Space Creation Glossary

Space

- A top-level container for content in Confluence. Every space has its own set of pages, permissions, and settings.

Page

- A document inside a space where content is created and edited. Pages can be nested into a hierarchy.

Child Page / Parent Page

- Pages form a tree structure: parent pages hold child pages beneath them. Used to organize content hierarchically.

Homepage

- The landing page of a space. Usually customized to provide navigation, instructions, or key info.

Global Permissions

- Permissions granted at the site level (e.g., who can create spaces, administer spaces).

Space Permissions

- Permissions that apply to an entire space (e.g., who can view, add, delete, or administer pages in that space).

Page Restrictions

- Permissions applied at the page level. Pages can be restricted to specific users/groups for view or edit.

Anonymous Access

- When a space or page is visible to people who aren't logged into Confluence (commonly used for public Knowledge Bases).

Page Tree

- The hierarchical navigation menu on the left side of a space showing all pages and their structure.

Templates (Page Templates - a page you copy/duplicate)

- Pre-defined page layouts that help maintain consistency (e.g., Meeting Notes, Project Plan).

Blueprints

- Special types of templates provided by Confluence for common use cases (e.g., Product Requirements, Decision log). Can be created by Atlassian, Confluence Admins, or Space Admins

Labels

- Tags added to pages to improve searchability and grouping of related content.

Space Categories

- Labels applied to spaces (not pages) that help group spaces together in the space directory.

Space Settings

- Admin panel for each space where you configure permissions, look & feel, content tools, and integrations.

Star space

- A bookmark for your space so that you can easily navigate to the space, helpful when you have access to many different spaces.

Watch Page

- Receive notifications when others edit this specific page that you watched.

Automation

- Created by Space Admins, allows for things to automagically happen in the background or with the press of a button if the automation rule is a manual trigger.

Content Tools

- A space admin feature for managing things like templates, trash, and export options.

Space Directory

- A list of all spaces in Confluence, often searchable and filterable by category.

Trash

- Deleted pages in a space go here; space admins can restore or permanently delete them.

Space Trash

- Deleted spaces go here; space admins can restore or permanently delete them.

Space Archive

- If you don't want to permanently delete a space, but don't want anyone to access, you have the option to archive a space. Note, this is a Confluence Premium/Enterprise feature only.

Macros

- Small add-ons you embed in a page to display dynamic content (e.g., Jira Work Items macro, Table of Contents, Page Tree).

Jira Link / Jira Macro

- Integration that lets you embed live Jira work items, boards, and reports into Confluence pages.

Public Links (if enabled)

- A feature that allows sharing a Confluence page with external users without giving them full access.

Live Doc

- Like a Confluence page, but no publishing is required. Multiple people can collaborate at the same time without having to publish or refresh to get changes. You lose version history.

Whiteboard

- A collaborative whiteboard that allows you to visually communicate with your team.

Database

- A table to store data more dynamically than you would with a regular table in Confluence

Setting Up a Confluence Space for Software Teams Run Book

Home (Space Homepage)
- Team Overview
  - About the Team
  - Roles & Responsibilities
  - Contact Information / Slack channel links
  - Onboarding Guide
- Goals & Strategy
  - OKRs / KPIs
  - Roadmap (linked from Jira or manually updated)
  - Product Vision / Charter
- Projects & Work
  - Active Projects
    - Project 1 Overview
    - Project 1 Specs / Requirements
    - Project 1 Decisions
  - Archived Projects
  - Backlog Ideas (link to Jira Product Discovery or Idea pages)
- Documentation
  - How-to Guides
  - Architecture & Systems
    - System Diagrams
    - APIs
    - Dependencies
  - Tools & Processes
- Meetings & Rituals
  - Sprint Planning Notes
  - Retrospectives
  - Daily Standup Notes (optional, if team uses them)
  - Stakeholder Updates
- Announcements
  - Release Notes
  - Change Logs
  - Team Updates
- Resources
  - Templates
  - Glossary
  - External References

Software Teams Templates

Project / Feature Pages

- Project Overview Template: Goals, stakeholders, timeline, Jira links.
- Product Requirements Template: Problem statement, solution design, acceptance criteria, Jira epics/stories linked.
- Decision Log Template: Context, decision taken, alternatives considered, date, owner.

Agile Rituals

- Sprint Planning Template: Sprint goal, committed stories, capacity.
- Retrospective Template: What went well, what didn't, action items.
- Standup Notes Template (if documented): Yesterday, today, blockers.

Knowledge Sharing

- How-to Guide Template: Problem, steps, expected outcome, troubleshooting.
- Architecture Doc Template: Diagram (use Draw.io / Gliffy macro), description, key contacts.
- Onboarding Checklist Template: Accounts, tools, documentation links, mentors.

Communication

- Release Notes Template: Version, features, fixes, screenshots, links.
- Team Update Template: Weekly/monthly updates, highlights, blockers.

Keep hierarchy shallow → avoid burying pages more than 3 levels deep.
Make the homepage a launchpad → use macros like "Children Display" or "Page Tree" plus quick links to Jira boards.
Use labels → tag pages with meeting-notes, decision, architecture to improve searchability.
Link to Jira → embed live Jira work items, roadmaps, or boards inside project pages.
Automate with templates → encourage consistency for meetings, projects, and documentation.
Archive ruthlessly → move old projects into an "Archive" section to keep the space clean.

Setting Up a Confluence Space for HR Teams Run Book

Home (Space Homepage)
- About HR
  - Mission & Values
  - Team Directory & Roles
  - Contact HR / Support Channels
- Policies & Guidelines
  - Employee Handbook
  - Code of Conduct
  - Workplace Policies (Leave, Travel, Remote work)
  - Compliance & Legal
- Onboarding & Offboarding
  - New Hire Welcome Guide
  - Onboarding Checklist
  - Training & Learning Resources
  - Offboarding Checklist
- Compensation & Benefits
  - Salary Bands / Pay Structure (internal-only)
  - Benefits Overview
  - Health & Wellness Programs
  - Perks & Discounts
- HR Processes
  - Recruiting & Hiring
    - Job Descriptions
    - Interview Guides
    - Referral Program
  - Performance Reviews
  - Promotions & Career Development
  - Time-off Requests & Holidays
- Employee Relations
  - Feedback Channels
  - Grievance Process
  - DEI (Diversity, Equity & Inclusion) Initiatives
- Announcements
  - Company Updates
  - HR Newsletters
  - Policy Changes
- Resources
  - HR Templates & Forms
  - Training Materials
  - FAQs
  - External Resources (links to payroll system, benefits portal, etc.)

HR Teams Templates

Employee Lifecycle
New Hire Onboarding Template: Welcome, accounts setup, training checklist, buddy assignment.
Exit Interview Template: Questions, feedback summary, action items.
Training Session Template: Agenda, materials, attendance, key takeaways.

Policies & Documentation
Policy Page Template: Purpose, scope, policy details, effective date, owner, review date.
HR Process Template: Step-by-step workflow, responsible parties, links to forms/systems.
FAQ Template: Question, answer, related resources.

Communication
Announcement Template: Subject, summary, details, effective date, next steps.
Newsletter Template: Highlights, HR spotlight, upcoming events, resources.

People Management
Performance Review Template: Goals, feedback, ratings, next steps.
Career Development Plan Template: Strengths, growth areas, milestones, training.

Confluence Spaces Creating Mistakes To Avoid

Creating Too Many Spaces

- Spinning up a new space for every small project or team.
- Leads to fragmentation, duplication, and hard-to-find content.
- Instead: consolidate into fewer, well-structured spaces with clear page hierarchies.

No Clear Purpose for the Space

- Starting a space without defining its goal or audience.
- Users don't know if it's for documentation, collaboration, or policies.
- Before creating, answer: "Who will use this space and why?"

Poor Page Hierarchy / Flat Structure

- Dumping all pages at the root with no logical organization.
- Makes navigation difficult and discourages content contributions.
- Use a top-level "Home" page with clear sections (Knowledge, Policies, Projects, etc.).

Not Using Templates or Standards

- Everyone creates content differently (free-text pages, no structure).
- Knowledge base articles look inconsistent.
- Provide templates for meeting notes, policies, incident reports, FAQs, etc.

Ignoring Permissions & Access Controls

- Giving everyone full admin rights → chaos.
- Locking down everything → no collaboration.
- Balance permissions: admins for governance, contributors for team content, and read-only for others.

Forgetting About Labels & Metadata

- Content is untagged → search becomes useless.
- Users can't filter by topic.
- Encourage consistent labeling (e.g., "incident", "policy", "how-to").

Not Setting a Space Homepage

- Default homepage looks empty/uninviting.
- Users can't find key info at a glance.
- Design a landing page with navigation, search bar, and quick links.

Duplicating Jira/Other Tool Data

- Copy-pasting Jira work items or roadmaps manually → goes stale.
- Instead: embed Jira filters, roadmaps, or Trello boards dynamically in Confluence.

No Governance or Archiving Strategy

- Old content piles up → search results become noisy.
- Establish content lifecycle rules: review quarterly, archive outdated pages, and mark drafts.

Not Considering End-User Experience

- Spaces built for admins, not for the people who actually use them.
- End-users struggle to find FAQs, guides, or policies.
- Think of your space like a website: make it easy to browse and search.

Confluence Blueprint Creation Guide

Step 1: Understand What a Blueprint Is

- A blueprint is different from a normal template but most use template and blueprint interchangeably. Blueprints can be space specific, or global.
- Templates = reusable page structures. Any page can be a "template" because you just duplicate the page.
- Blueprints = templates + extra functionality (like automatic indexing, prompts, or page creation wizards).
- Examples Atlassian provides: Meeting Notes, Product Requirements, Decision Logs.

Use a blueprint when:

- You want consistency across multiple spaces
- You need a create-page wizard that asks questions before building the page
- You want to auto-organize pages (e.g., all meeting notes under a "Meetings" parent page)

Step 2: Decide What Content Needs a Blueprint

Ask:

- Do you want a team-wide standard format (e.g., Incident Postmortem, Onboarding Checklist, Project Charter)?
- Do I want my team to follow the same format every time they create this type of page?
- Should metadata (labels, dates, authors) be added automatically?

Step 3: Prepare the Template Content

- Go to Space Settings → Look and Feel → Templates
- Click on Create a new template
- Draft the layout of the page you want standardized. This is basically like creating a normal page.
- Use headings, tables, and macros (status, panels, task lists).
- Example (Incident Postmortem Blueprint):
  - Summary
  - Timeline of Events
  - Root Cause
  - Lessons Learned
  - Action Items (task list macro)

Publish → users can now select it in the "Create" page dialog.

Step 4: Test It

Create a test page, but this time search for the blueprint name in the template browser search.
Check if metadata is applied, parent page works, and structure looks right.
Get feedback from end-users.

Confluence Space Permissions Guide

1. What Are Space Permissions?

- Space permissions control who can access and do what in a Confluence space.
- They apply at the space level (not page level — that's done with page restrictions).
- Permissions can be assigned to groups or individual users.
- Each permission grants a specific action, like viewing, adding pages, exporting, or administering the space.

2. Types of Permissions

Here's a breakdown of the main permission categories:

Permission Type / Role	What It Allows	Example Use Case
Admin	Can manage everything in the space	Folks that can rename space, create automation rules, create blue prints
Manager	Can manage people and content, but no space settings	Power user, but maybe not technical enough to change space settings.
Collaborator	Can add and edit content	Folks that need to interact by creating and editing content.
Viewer	Can view and comment on content	Folks that should only be able to see content, but not create anything or edit anything.
Custom Access	Defined by Confluence Admin	Custom role that can be tweaked to contain over 20 different permissions.

3. Where to Configure Space Permissions

Go to the space → Space Settings → Users
To add a new person, click on Add people on the right side.
In the pop up window, search for a user or group and assign to one of the available groups.
Click on save.

To edit an existing user or groups role, search for the user/group in the User's page and then click on the drop down for the role.

You can also remove a user/group from the User's page.

Confluence Page Permissions Guide

1. What Are Page Restrictions?

- Page restrictions control who can view or edit an individual page.
- They override space permissions as they are a "subset" from the space permission.
- Useful when you need to share a space broadly, but lock down specific sensitive pages.
- Two main types:
  - View Restriction → Who can see the page.
  - Edit Restriction → Who can make changes to the page.

2. When Should You Use Restrictions?

- Sensitive information (e.g., salary policies in HR space).
- Drafts you don't want visible yet.
- Confidential project plans limited to a subset of a team.
- Postmortems or incident reports with security details. Don't use them as your main permission system — that's what space permissions are for.

3. How to Add Restrictions to a Page

- Step 1: Open the Restrictions Dialog
  - Go to the page you want to restrict.
  - In the the Share button (will have a lock).
  - Select the users that you want to "share" the soon to be locked page.

- Step 2: Choose Restriction Type
  - You'll see three options:
  - No restrictions (anyone with space permissions can view/edit).
  - Editing restricted (everyone can view, but only selected users/groups can edit).
  - Viewing and editing restricted (only selected users/groups can view/edit).

- Step 3: Verify
  - Log in as a test user or use "View as user" (if available in your Confluence version).
  - Confirm the right people can see/edit the page.
  - Alternatively, have someone that should have access to the page confirm they can see the page
  - Have someone that shouldn't be able to access the page confirm they can't see the page.

4. Managing Restrictions

- Inherited Restrictions:
- If a parent page is restricted, child pages may inherit those restrictions.
- Example: If you lock a top-level "HR" page to HR staff only, all subpages inherit that restriction unless changed.
- Checking Existing Restrictions:
  - Go to the Share Button → Restrictions to see who has access.
  - Or use the Page Information option to see restrictions history.
- Removing Restrictions:
  - Open the restrictions dialog and click the x next to a user/group.
  - Switch the page back to No Restrictions to reset it.

ATLASSIAN CLOUD ADMINISTRATION

User Management Billing Guide

Al users across your Atlassian cloud products are managed in admin.atlassian.com. Here, you grant licenses, grant product access, manage groups, and so much more. This guide is designed to help you keep your costs down by knowing how to audit your users. So many of the teams I work with have dozens and sometimes hundreds of inactive users that are costing them thousands of dollars.

Step 1: Invited Users

- When you invite users, if you grant them access to a product (Jira/Confluence), they will count against your billing.
- Make sure you encourage users to actually log into the tools, otherwise remove them because you'll be paying for every month they don't actually log in.

Step 2: Last Log In Date

- Some users log into Jira once or twice and then forget about it. As a Site/ORG admin, you should audit every user and watch for the last time they logged into either Jira or Confluence.
- Remove product access/license from any user that hasn't used the product within the last 90 days. This will help free up licenses as opposed to buying more licenses.
- If you set up your App Access correctly, unactivated users can easily retrieve their licenses after you revoke them.
- Don't delete or suspend the user, just remove their product access/license.

Step 3: App Access Review

- Atlassian makes it easy for users to get licenses. In fact, they make a little too easy. As a result, most organizations have very lose restrictions and simply allow anyone that a URL to a Jira work item or a Confluence space to automatically join and get a license.
- Don't do this unless you want to very quickly inflate your Atlassian bill
- At a minimum, you should only allow for Any Domain users to be JSM Customers (if you are using JSM). All other apps should be turned off.
- For your companies main domain(s), you can configure this however you want, but I would recommend, if your goal is to keep costs down, to require all user requests to be reviewed by a Site/ORG admin before the actual license is granted.

Step 4: Cross Instance Audit

- If you aren't an Enterprise customer and you have multiple Jira's / Confluence's, then audit any users that shouldn't be access the extra Jira/Confluence. Standard and Premium subscribers have to pay for each user, across each product. So, if someone needs to access 2 or more Jira's, you'll pay 2 or more times for that specific individual.

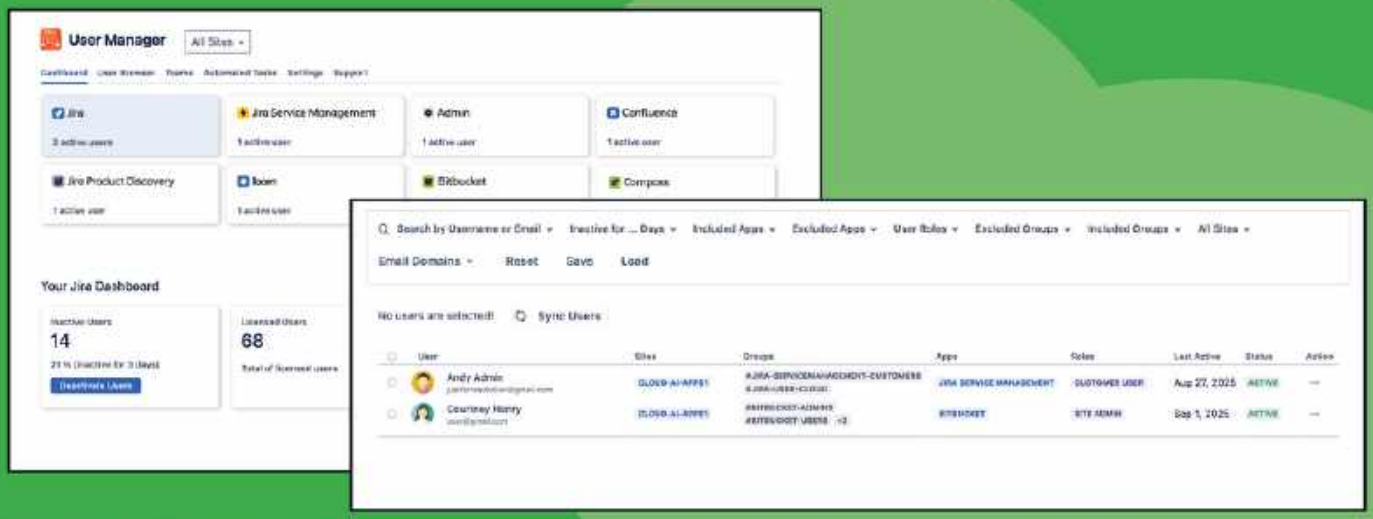
User Management &
License Optimizer

re:solution

SMART USER MANAGEMENT, BIG SAVINGS

Cut Atlassian license spend by automating user lifecycle tasks so you're only paying for who actually needs access. Streamline deactivations, control license usage across your organization, and keep admin effort minimal.

- Run bulk deactivations in a few clicks
- Automate group changes and user lifecycle tasks
- Identify inactive users instantly through the dashboard
- Unified management across all Atlassian apps and sites with one install
- See immediate savings of 15-30% on license costs
- Built on Forge for unlimited scale and security
- Browse all users with advanced filters and real-time sync



APP INFO

BOOK A DEMO

www.resolution.de

Understanding Atlassian Administrators Guide

	ORG ADMIN		Ultimate administrator. Oversees all of the Atlassian cloud products and ultimately responsible for billing. Is an admin for each and every site under the org. Can manage all users across all sites.
	SITE ADMIN		Very similar to the org admin, but with two key differences. The site admin doesn't oversee billing, and is the site admin for only the specific site. Can manage users only for their specific site.
	administrator		Below the Site admin, the administrator is a technical administrator for both Confluence and Jira. Cannot manage any users.
Technical administrator for Jira only. Can manage workflows, fields, etc.	Jira Admin	Confluence Admin	Technical administrator for Confluence only. Can manage spaces, global automations, global templates, etc.
Administrator of a specific space. If the space is company managed, they can rename the space. If it's team managed, they can modify workflows, fields, etc.	Space Admin	Space Admin	Administrator for a specific space. Can add users, groups, create local automations, local templates, etc.

Atlassian Guard Shadow IT Guide

When you are an Atlassian Guard subscriber, you pay for each user's ability to use their email (domain) to SSO/Authenticate with your Atlassian Cloud products. Atlassian Guard is a separate subscription and is in addition to your Jira/Confluence/JSM subscription. If you are an Enterprise subscriber, Atlassian Guard is included in your annual pricing, except for one special condition that I wanted to cover here.

This technically applies for all Atlassian Guard subscribers, but the effects are harsher for Enterprise subscribers. With Enterprise, you pay for once for each user and as long as they access an Enterprise Jira/Confluence/JSM, then that's it, you don't pay anything else. The problem is IF a user accesses a non Enterprise Jira/Confluence/JSM.

If a user goes to Atlassian.com and "buys" their own Jira/Confluence (remember these products are technically free) AND they don't access one of your Enterprise products, but they used their company email, then Atlassian is going to count this user as a billable user and will send you a separate Atlassian Guard bill when you renew. Luckily, Atlassian is going to let you know that the user created a "discovered" product (more on this later in the book). The only way to fix this is for the user to remove their product or you pull them into one of your Enterprise Jira/Confluence/JSM. You also have the option to upgrade their Jira/Confluence to an official Enterprise product under your Org.

A second scenario is if a user accesses a different Jira/Confluence. Maybe a vendor or consultant invited one of your users to use their Jira/Confluence. If they are using their company email which is tied to your Atlassian Guard AND they don't normally access one of your Enterprise products, then you'll also see a bump in your Atlassian Guard billing.

A third scenario is if the user accesses a different Atlassian product like Trello or Bitbucket. Again, if they don't normally access one of your official Enterprise products, then Atlassian is going to happily bill you for these users.

I find that Atlassian doesn't really give us any way to prevent this from happening, but they do allow us to see who the individuals are. I find that it's best to simply communicate with these individuals and make them aware of the hidden costs from Atlassian. 99% of the time, people just don't know how Atlassian works and/or bills when customers have Atlassian Guard.

I hope this guide helps you save a few (hundred) dollars.

Discovered Products	Whenever a user with your domain “purchases” their own Atlassian cloud product, it will show up here. You don’t technically pay for these products as the person that purchased the product will need to provide billing details, but if you have Atlassian Guard, you will need to pay for their Guard license assuming they don’t already have one (by accessing one of your enterprise products only).
User API Tokens	Depending on your Authentication Policy, users may be able to get their own API tokens. Any tokens generated by any of your users will show up here. You can revoke any license and/or update your Authentication policy to prevent users from getting an API token.
IP Allowed List	If you want to make your Atlassian Cloud a little more secure, you can specify which IP’s are allowed to access your cloud products. This becomes a lot harder to manage if your users aren’t using a VPN to get on to your network.
Audit Log	If you have Atlassian Guard, you can see a log of every interaction by every user. This is a very powerful log and I would recommend making sure you have guard for this feature (and SSO/Authentication).
Authentication Policy	Controls the rules security rules that will govern a specific user (or groups of users). Most importantly, you can set how long a user stays logged in for, how they log in, and if they can get an API token.
Insights	See how your users are using your Atlassian cloud products. This is very helpful to understand how many actual active users you have versus how many licensed users you have.
User Counts	The ONLY way to truly see how many licenses your organization is consuming. Keep a watchful eye on this. Atlassian will alert you when you hit 80% of your license counts.

Marketplace App Evaluation Framework

1. Define the Business Need

- Problem statement: What gap are you solving? (e.g., “We need better time tracking,” or “We need advanced reporting.”)
- Alternatives: Can Jira natively handle this, or with automation/custom fields/workflows?
- Impact: Who benefits? (admins, end-users, executives?)
- If you can’t clearly define the business value, don’t install the app.

2. Evaluate Vendor Reputation

- Vendor profile: Are they an Atlassian Platinum Partner or a small shop?
- App age: How long has the app been in the Marketplace?
- Active installs: How many customers use it?
- Reviews: What do other admins say about support and reliability?
- Update frequency: Is the app actively maintained and compatible with the latest Jira versions?
- Look for vendors with strong Marketplace presence and active development.

- **3. Security & Compliance Checks**
- Atlassian Cloud Fortified? → Has it passed Atlassian’s enhanced security checks?
- Data residency: Where is customer data stored?
- Data handling: Does the app store data outside Atlassian? (Check the privacy policy.)
- SOC2 / ISO27001 compliance if relevant for your company.
- Permissions requested: Does the app request more access than it should need?
- Work with your security/compliance team if data is sensitive.

4. Functionality Fit

- Core features: Does it meet your exact use case?
- Ease of use: Is it intuitive for non-admins?
- Ask for a demo or free trial to test in a sandbox.

5. Performance & Reliability

- Impact on Jira performance: Any reports of slowdowns?
- Cloud vs. Data Center differences: Same features across hosting models?
- Uptime & SLAs: Does the vendor provide status pages or uptime guarantees?

6. Cost & Licensing Model

- Pricing tiers: Does cost scale with user count? (Most Marketplace apps do.)
- Hidden costs: Training, consulting, admin overhead.
- New Pricing Models: What functionality do you get on Standard vs Advanced?
- ROI check: Does the app save enough time/money to justify cost?
- Calculate annual cost at your full license tier (not just current user count).

7. Support & Documentation

- Documentation quality: Is it clear, up to date, and easy to follow?
- Support SLAs: How quickly does the vendor respond?
- Community presence: Are there active community/forum discussions?
- Choose vendors with strong documentation and responsive support.

ALWAYS TEST IN A SANDBOX ENVIRONMENT FIRST!!!!

Agile Project Management

Silver
Solution Partner

CLOUD SECURITY PARTICIPANT
CLOUD FORTIFIED

catapult LABS

Collaboration Apps for Remote & Agile Teams

Agile Rituals That Move the Needle

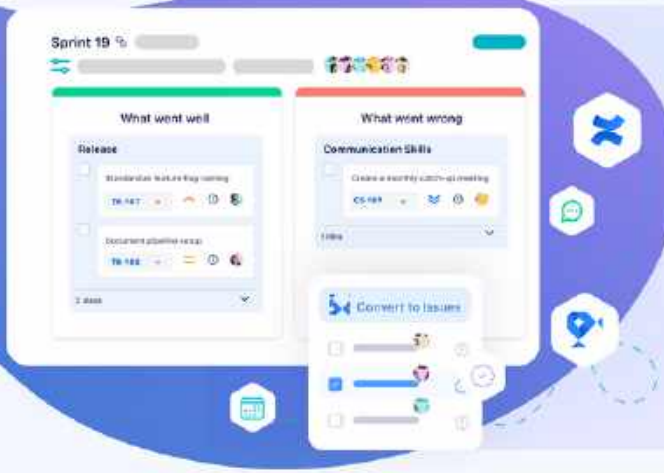
Our tools build momentum towards a culture that empowers developers to do their best work

Better Metrics

Enhanced Collaboration

Cost Savings

Continuous Improvement



Retrospectives

Run Agile Retrospectives in Jira & Confluence

Fully custom, real-time & async sessions

Turn action items into Jira issues & Confluence tasks

ScrumPoker

Estimate software tasks playing Planning Poker

StandBot

Async stand-ups in Slack, linked with Jira

GitLinker

Branch, commit & pull request manager for Jira

Try Our Apps Today!

Scrum vs. Kanban Comparison Table

Aspect	Scrum	Kanban
Core Philosophy	**Iterative Development.** Work is done in fixed-length cycles (**Sprints**) to deliver a "Potentially Shippable Increment."	**Continuous Flow.** Focus is on visualizing the workflow, limiting Work in Progress (**WIP**), and maximizing efficiency.
Cadence & Rhythm	**Time-boxed and regular.** Sprints are typically 1-4 weeks (ideally 2) long, providing a consistent rhythm for planning and review.	**Continuous and event-driven.** There are no prescribed iterations. Work is pulled into the system as team capacity becomes available.
Roles	**Prescribed Roles.** Requires a **Product Owner** (manages the backlog), a **Scrum Master** (facilitates the process), and the **Development Team** (does the work).	**No Prescribed Roles.** It's designed to be applied to your existing team structure. Roles may emerge but are not required.
Key Metrics	**Velocity** (how much work is completed per Sprint) and **Burndown Charts** (tracking progress toward the Sprint Goal).	**Lead Time** (total time from request to completion), **Cycle Time** (time from when work begins to completion), and **Throughput** (items completed per unit of time).
Handling Change	Changes are discouraged during a Sprint to protect the Sprint Goal. New priorities are addressed in the next Sprint Planning.	Changes can be made at any time by re-prioritizing items in the backlog. New work can be pulled in as soon as capacity allows.
Meetings & Ceremonies	**Prescribed and required.** Includes **Sprint Planning**, **Daily Scrum**, **Sprint Review**, and **Sprint Retrospective**.	**No prescribed meetings.** Teams often adopt meetings as needed, such as a daily stand-up, replenishment meetings, and service delivery reviews.
Release Cycle	Releases typically happen at the end of a Sprint, but can be done at any time. The focus is on a regular, predictable delivery of value.	**Continuous Delivery.** A new item can be released as soon as it is finished. There is no set schedule for releases.
Boards	The **Scrum Board** is reset at the end of every Sprint.	The **Kanban Board** is persistent. Columns represent workflow states, and cards move across the board continuously.

Scrum Glossary

Scrum Team

- **Scrum Team:** A cohesive unit of professionals focused on one objective at a time, the Product Goal. It consists of one Scrum Master, one Product Owner, and Developers.
- **Product Owner**: The person accountable for maximizing the value of the product resulting from the work of the Scrum Team. They are primarily responsible for managing the Product Backlog.
- **Scrum Master**: The person accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.
- **Developers:** The people in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint.

Scrum Events

- **Sprint**: A fixed-length event of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happens within Sprints.
- **Sprint Planning:** The event that initiates the Sprint by laying out the work to be performed. The plan is created by the collaborative work of the entire Scrum Team.
- **Daily Scrum:** A 15-minute event for the Developers of the Scrum Team to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary.
- **Sprint Review**: The second to last event of the Sprint, held to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.
- **Sprint Retrospective**: The final event of the Sprint. The purpose is to plan ways to increase quality and effectiveness. The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done.

Scrum Artifacts

- **Product Backlog**: An emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team.
- **Sprint Backlog:** A plan by and for the Developers. It is composed of the Sprint Goal (the "why"), the set of Product Backlog items selected for the Sprint (the "what"), as well as an actionable plan for delivering the Increment (the "how"). Jira only has a single “backlog”
- **Increment:** A concrete stepping stone toward the Product Goal. Each Increment is an additive to all prior Increments and thoroughly verified, ensuring that all Increments work together. To be considered a part of an Increment, the work must meet the Definition of Done.

Other Key Terms

- **Product Goal:** Describes a future state of the product which can serve as a target for the Scrum Team to plan against. The Product Goal is in the Product Backlog.
- **Sprint Goal:** The single objective for the Sprint. It is an objective set by the Scrum Team and provides focus and coherence during the Sprint. The Sprint Goal is part of the Sprint Backlog.
- **Definition of Done (DoD):** A formal description of the state of the Increment when it meets the quality measures required for the product. When a Product Backlog item meets the Definition of Done, an Increment is born.

Kanban Glossary

Kanban Board Elements

- **Kanban Board:** A visual tool used to manage and track work as it moves through a process. Each column on the board represents a stage in the workflow.
- **Work Item (Card):** A single unit of work that is being tracked. In its simplest form, a card on a Kanban board represents a task. Cards contain information about the work, such as a description, owner, and due date.
- **Columns (Workflow States):** The vertical lanes on a Kanban board that represent the different stages of the workflow. Common examples include "To Do," "In Progress," and "Done," but these are customized to fit the team's actual process.
- **Swimlanes:** Horizontal lanes on a Kanban board used to categorize work items. Swimlanes can be used to separate different types of work, priorities (like an "expedite" lane), by Epic, by assignee, or custom queries.

Key Metrics

- **Work in Progress (WIP):** The total number of work items that have been started but not yet completed. This is the work that is currently "in the system."
- **WIP Limits:** A constraint placed on a column (workflow state) or the entire system to limit the number of work items that can be active at one time. This is a crucial mechanism for creating a pull system.
- **Lead Time:** The total time a work item spends in the system, from the moment it is requested (committed to) until it is delivered.
- **Cycle Time:** The time it takes to complete a work item from the moment work actively begins until it is finished. Cycle time is a subset of lead time.
- **Throughput:** The number of work items completed per unit of time (e.g., tasks per week). This is a measure of the team's delivery rate.

Kanban Events (Feedback Loops)

While Kanban does not prescribe mandatory meetings like Scrum, it encourages regular feedback loops to improve the process.

- **Kanban Meeting (Stand-up):** A short, daily team meeting held in front of the Kanban board. The focus is on the flow of work, discussing blocked items and what can be done to move work to the next stage.
- **Replenishment Meeting:** A meeting held to review items in the backlog (or "input queue") and decide which ones to select next for the team to work on. This is the point where the team "pulls" new work into the system.
- **Service Delivery Review:** A meeting focused on reviewing the team's performance against their service-level expectations (SLEs). The team analyzes metrics like lead time and throughput to see if they are meeting their delivery goals.

Agile Roles & Responsibility Guide

Scrum Roles
The Scrum framework prescribes three specific, accountable roles within a cohesive unit called the Scrum Team. These roles are not job titles but represent a set of responsibilities.

1. Product Owner
The Product Owner is the voice of the customer and is accountable for maximizing the value of the product created by the team.

Key Responsibilities:
- Managing the Product Backlog: This includes creating, clearly communicating, and ordering backlog items to best achieve goals and missions.
- Defining the Product Goal: The Product Owner is responsible for developing and explicitly communicating a long-term objective for the product.
- Stakeholder Management: Works with stakeholders to understand their needs and ensure the Product Backlog reflects them.
- Guiding the Development: Makes the final decision on what the team will work on during a Sprint and is the only person who can cancel a Sprint if the Sprint Goal becomes obsolete.

2. Scrum Master
The Scrum Master is a servant-leader responsible for ensuring the Scrum framework is understood and enacted. They serve the Product Owner, the Developers, and the organization as a whole.

Key Responsibilities:
- Facilitating Scrum Events: Ensures that Sprint Planning, Daily Scrums, Sprint Reviews, and Sprint Retrospectives take place and are positive, productive, and kept within their timeboxes.
- Removing Impediments: Helps remove blockers to the team's progress.
- Coaching the Team: Coaches the team members in self-management and cross-functionality.
- Promoting Scrum Practices: Helps the team and the wider organization adopt and understand Scrum theory, practices, rules, and values.

3. Developers
The Developers are the professionals within the Scrum Team who are committed to creating any aspect of a usable product Increment each Sprint. For teams that don't do software development, developers are any member of the team that does the actual work.

Key Responsibilities:
- Creating the Increment: Performs the hands-on work of designing, building, and testing the product.
- Managing the Sprint Backlog: Creates a plan for the Sprint (the Sprint Backlog) and adapts it each day to meet the Sprint Goal.
- Ensuring Quality: Instills quality by adhering to a Definition of Done.
- Holding Each Other Accountable: As professionals, Developers are expected to manage their own work and collaborate effectively to achieve the Sprint Goal.

Kanban Roles
Kanban is a method, not a framework, and is less prescriptive than Scrum. It does not require specific roles and is designed to be applied to your existing organizational structure. The focus is on the work and the flow, not on predefined job titles.

However, over time, some service-oriented roles often emerge to help manage the flow.

1. Service Request Manager (or "Flow Manager")
This is an emergent role that focuses on understanding customer needs and managing the flow of work from request to commitment. This person often helps facilitate the Replenishment Meeting. This role is sometimes compared to the Product Owner in Scrum.

Key Responsibilities:
Managing Customer Expectations: Acts as a liaison between the customer and the delivery team.
Prioritizing the Backlog: Helps the team decide what work to pull in next based on business value, risk, and other factors.

2. Service Delivery Manager (or "Flow Master")
This role is focused on improving the efficiency and effectiveness of the team's workflow. This role is sometimes compared to the Scrum Master.

Key Responsibilities:
- Facilitating Kanban Meetings: Ensures feedback loops like the daily Kanban Meeting and Service Delivery Review are effective.
- Analyzing Flow Metrics: Tracks metrics like Lead Time, Cycle Time, and Throughput to identify bottlenecks and opportunities for improvement.
- Protecting the Flow: Helps the team manage blockers and improve their process policies to create a smoother, more predictable delivery system.

Scrum Master Playbook

Here is a step-by-step guide for a Scrum Master's Playbook, designed to help a Scrum Master navigate their role from joining a new team to fostering continuous improvement.

Step 1: The First 30 Days - Observe and Connect
Your initial goal is to understand the landscape without making drastic changes. The focus is on observation, relationship-building, and assessing the current state of agility.

- **Meet the Team:** Schedule one-on-one meetings with every member of the Scrum Team (Product Owner, Developers) and key stakeholders. Ask open-ended questions about their roles, challenges, and what they think is working well.
- **Observe the Events:** Silently observe each Scrum event (Planning, Daily Scrum, Review, Retrospective). Take notes on how they are run, the level of engagement, and any immediate anti-patterns you notice.
- **Understand the Product:** Spend time with the Product Owner to understand the Product Goal, the state of the Product Backlog, and the key business objectives.
- **Assess Agile Maturity:** Create a baseline understanding of the team's grasp of Scrum principles. Note their strengths and identify the top 1-2 areas that could benefit from coaching.

Step 2: Facilitating Your First Full Sprint Cycle
Now you move from a passive observer to an active facilitator. Your goal is to ensure each event is productive and stays true to its purpose as defined in the Scrum Guide.

- **Sprint Planning:**
  - **Before**: Work with the Product Owner to ensure the Product Backlog is in good shape and a potential business objective for the Sprint is ready.
  - **During**: Guide the team in crafting a clear Sprint Goal. Ensure the Developers have enough information to select the work and create a plan (the Sprint Backlog).
  - **After**: Make sure the Sprint Goal and Sprint Backlog are visible to everyone.
- **Daily Scrum:**
  - **During**: Ensure the meeting starts on time and stays within the 15-minute timebox. Coach the Developers to focus on their progress toward the Sprint Goal and to identify any impediments. It is their meeting; you are primarily there to ensure it happens correctly.
- **Sprint Review:**
  - **Before**: Encourage the team to prepare for a "show, not tell" session.
  - **During**: Facilitate a collaborative working session, not just a demo. Actively encourage feedback from stakeholders and guide the conversation back to the Product Goal. Every team member should present something.
- **Sprint Retrospective:**
  - **During**: Use an engaging format (e.g., "4Ls," "Sailboat") to guide a constructive conversation about what went well and what could be improved.
  - **After**: Ensure the team's improvement items are actionable and visible, perhaps by adding them to the next Sprint Backlog.

Step 3: Driving Continuous Improvement

With the basics in place, your focus shifts to a higher level: coaching, removing systemic impediments, and elevating the team's performance.

- **Impediment Management:**
  - Create and maintain an impediment log that is visible to the team.
  - Coach the team to solve their own problems first, but be ready to escalate organizational impediments they cannot resolve themselves.
- **Coaching:**
  - **For Individuals:** Provide one-on-one coaching to team members on Agile principles and practices.
  - **For the Team:** Coach the team toward self-management, encouraging them to take ownership of their process and decisions.
- **Using Metrics Wisely:**
  - Introduce metrics like burndown charts or cycle time not as performance reports for management, but as tools for the team to inspect and adapt their own process.
  - Facilitate conversations around these metrics during the Sprint Retrospective to identify patterns and opportunities for improvement.

Work Type Hierarchy Breakdown

Level 1: Theme
A **Theme** is a high-level strategic objective that drives the creation of various product features and initiatives. It represents a broad area of focus for the business and typically spans multiple quarters or even years.

- **Purpose**: To align the organization around a major business goal.
- **Example**: "Improve Customer Retention in 2025"

Level 2: Initiative
An **Initiative** is a collection of large-scale projects or Epics that work together to achieve a specific Theme. Initiatives are often managed at the portfolio level and may involve multiple teams.

- **Purpose**: To break down a strategic theme into more manageable, actionable parts.
- **Example**: "Launch a Customer Loyalty Program"

Level 3: Epic
An **Epic** is a large body of work that can be broken down into a number of smaller, deliverable pieces of functionality. It is a feature or project that is too big to be completed in a single sprint.

- **Purpose**: To group related user stories together into a complete feature.
- **Example**: "Develop a Points-Based Rewards System for Customers"

Level 4: User Story
A **User Story** is a small, self-contained unit of development work that delivers a specific piece of value to an end-user. It is typically written from the user's perspective.

- **Purpose**: To describe a software feature in a way that is understandable to both business and technical team members.
- **Example**: "As a customer, I want to see my current points balance on my account dashboard so that I can track my rewards."

Level 5: Task / Sub-task
A **Task or Sub-task is** a specific, actionable piece of work that needs to be done to complete a User Story. These are the granular activities that developers work on day-to-day.

- **Purpose**: To break down a user story into the technical steps required for its implementation.
- **Example**:
  - "Create a new component for the account dashboard to display the points balance."
  - "Develop an API endpoint to fetch the user's points data from the database."

Backlog Grooming Guide

Backlog Refinement is a continuous process where the Product Owner and the Developers collaborate to ensure the Product Backlog is in good shape. The goal is to keep the backlog clean, well-understood, and prioritized so that the team is ready for upcoming Sprints. It's not a formal Scrum event but an essential ongoing activity.

Key Activities in Backlog Refinement

The core of refinement involves making sure the items at the top of the backlog are ready for development. This is often remembered by the acronym **DEEP**: **D**etailed appropriately, **E**stimated, **E**mergent, and **P**rioritized.

- **Reviewing and Clarifying:** The team reviews upcoming Product Backlog Items (PBIs) to ensure everyone has a shared understanding of the work.
- **Adding Detail:** The Product Owner adds details, acceptance criteria, and mockups to user stories.
- **Estimating Stories:** The Developers add size estimates (using story points, t-shirt sizes, etc.) to the PBIs. This helps in forecasting and planning.
- **Breaking Down Large Items:** Large items (Epics) are broken down into smaller, more manageable user stories that can be completed in a single Sprint.
- **Re-Prioritizing:** Based on new information and feedback, the Product Owner may re-order the backlog to ensure the most valuable work is always at the top.

Who Participates?

- **The Product Owner:** Leads the meeting, presents the backlog items, and answers questions about business value and user needs.
- **The Developers:** Ask technical questions, provide insights on implementation, and give size estimates for the work.
- **The Scrum Master (Optional):** Can facilitate the meeting to ensure it stays on track and is productive.

How to Run an Effective Refinement Session

1. **Preparation (Product Owner):** Before the meeting, the Product Owner should prepare a handful of the highest-priority items to be discussed.
2. **The Meeting (The Whole Team):**
    - The Product Owner presents each item one by one.
    - The Developers ask questions to clarify requirements and technical feasibility.
    - The team collaborates to add acceptance criteria.
    - The Developers provide an estimate for each story.
3. **Outcome:** By the end of the session, the team should have a set of well-understood and estimated stories at the top of the backlog, ready to be pulled into the next Sprint Planning.

Consistent backlog refinement is one of the most effective ways to ensure your Sprint Planning meetings are fast and efficient and that the team maintains a steady, predictable pace of development.

Sprint Planning Playbook

Step 1: Before You Begin - The Preparation
A productive Sprint Planning meeting starts before the event itself. Without preparation, the meeting can become unfocused and inefficient.

- **Product Owner**: Comes prepared with an ordered Product Backlog that has been refined with the team. They should also have a business objective in mind for the upcoming Sprint.
- **Developers**: Should have an idea of their upcoming capacity (accounting for any time off) and be familiar with the work at the top of the backlog from refinement sessions.
- **Scrum Master:** Ensures the meeting is scheduled, attendees are invited, and the purpose of the event is clear to everyone.

Step 2: Topic One - Define the Sprint Goal (The "Why")
The first part of the meeting is dedicated to crafting a single, unifying objective for the Sprint.

- **Propose the Objective:** The Product Owner presents the business objective for the Sprint and highlights the Product Backlog Items (PBIs) that support it.
- **Collaborate and Negotiate:** The entire Scrum Team discusses the objective. The Developers provide input on the technical feasibility, and together, the team crafts a concise **Sprint Goal**.
- **Finalize the Goal:** The goal should be a short statement that provides focus and direction. It becomes the "why" for the Sprint.

Step 3: Topic Two - Select the Work (The "What")
With the Sprint Goal established, the Developers select the PBIs they forecast they can complete within the Sprint to achieve that goal.

- **Developers Pull the Work:** The Developers, not the Product Owner, pull items from the top of the Product Backlog.
- **Discuss Capacity:** The team discusses their available capacity, using past performance as a guide.
- **Ask Clarifying Questions:** The Developers ask the Product Owner detailed questions about the selected items to ensure a clear understanding before committing to the work.

Step 4: Topic Three - Create the Plan (The "How")
This is where the Developers create an actionable plan for how they will turn the selected PBIs into a finished Increment.

- **Break Down the Work:** The Developers decompose the selected PBIs into smaller, more granular tasks. This is the initial Sprint Backlog.
- **Organize the Plan:** The Developers may start to organize the work for the first few days of the Sprint. This plan is not rigid and can be adapted throughout the Sprint.
- **Confirm the Forecast:** After creating their initial plan, the Developers confirm their forecast of what they believe can be accomplished to meet the Sprint Goal.

Step 5: The Outcome - A Shared Commitment
By the end of Sprint Planning, the entire Scrum Team should be aligned and ready to start the Sprint.

- **The Sprint Goal is Finalized:** It is made visible to the team and stakeholders.
- **The Sprint Backlog is Created:** It consists of the Sprint Goal (the "why"), the selected PBIs (the "what"), and the Developers' plan (the "how").
- **The Team is Aligned:** Everyone on the Scrum Team agrees on the goal and is committed to working together to achieve it.

Sprint Planning Checklist

TIME	60-90 MINUTES FIRST	**CORDINATOR**	SCRUM MASTER
DATE	DAY OF THE SPRINT	**ATTENDEES**	SCRUM MASTER, DEVELPER , P.O., TECHNICAL LEADS/MANAGERS

1 PRE-PLANNING

- [ ] Refinement of new stories
- [ ] Rank backlog based on product owner guidance/priority
- [ ] Determine the sprint goal
- [ ] Determine team capacity account for: (PTO/Travel/Holidays)
- [ ] Tie-up previous sprint lose ends

2 PLANNING

- [ ] Create new sprint or close previous sprint
- [ ] Set dates and duration
- [ ] Set sprint goal
- [ ] Ensure all developers are present
- [ ] Use any rolled over stories as baseline for new sprint
- [ ] Estimate backlog items
- [ ] Assign items from backlog to developers
- [ ] Pull work items into sprint until 80% of capacity for each developer is
- [ ] met Review Insight to ensure sprint appropriately planned Review sprint
- [ ] capacity distribution
- [ ] Start sprint!

Sprint Execution Playbook

Step 1: Kick-Off and Focus

The first day of the Sprint sets the tone. The goal is to ensure the plan is visible and the team is aligned on its objective.

- **Visualize the Plan:** Immediately after Sprint Planning, ensure the Sprint Backlog is up-to-date and visible to the entire team on their physical or digital board.
- **Display the Sprint Goal:** Make the Sprint Goal prominent. It should be visible every day to serve as a constant reminder of the "why" behind the work.
- **Start the Work:** The Developers begin working on the items in the Sprint Backlog. A good practice is for the team to "swarm" or collaborate on the highest-priority item to get it done faster.

Step 2: The Daily Cadence

The heartbeat of the Sprint is the Daily Scrum. This is the team's opportunity to inspect progress and adapt their plan for the day.

- **Hold the Daily Scrum:** Every day, at the same time and place, the Developers meet for 15 minutes.
- **Inspect Progress:** The focus of the conversation should always be on the progress toward the Sprint Goal. Team members share what they accomplished, what they plan to do next, and any impediments they face.
- **Adapt the Plan:** Based on the discussion, the Developers update the Sprint Backlog and create a plan for the next 24 hours. The Daily Scrum is a planning meeting for the Developers, not a status report for managers.

Step 3: Managing Work and Quality

Throughout the Sprint, the team is responsible for managing their work and ensuring it meets the required quality standards.

- For the Developers:
  - **Collaborate**: Work together on tasks, use pair programming, and maintain a focus on getting items to "Done."
  - **Maintain Quality**: Adhere to the Definition of Done for all work. This includes writing tests, conducting code reviews, and ensuring the work is integrated.
  - **Update the Sprint Backlog:** As work is completed, the board should be updated in real-time to accurately reflect the state of the Sprint.
- **For the Scrum Master:**
  - **Remove Impediments:** Actively work to remove any blockers that the Developers have identified.
- **For the Product Owner:**
  - **Clarify Requirements:** Be available to answer questions and provide clarity on the Product Backlog Items as the Developers are working on them.

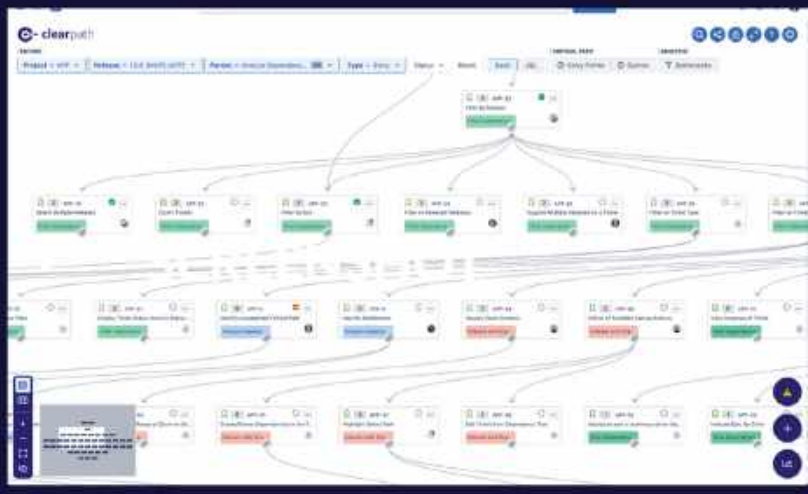
Step 4: Preparing for the Finish

As the Sprint nears its end, the team's focus shifts to completing the work and preparing to show what they've accomplished.

- **Focus on "Done":** The team's primary focus should be on completing any remaining "in-progress" items to meet their forecast. It's better to have fewer items completely "Done" than many items almost done.
- **Ensure the Increment is Ready:** Verify that all completed work is integrated and meets the Definition of Done, resulting in a potentially releasable Increment.
- **Prepare for the Sprint Review:** The team prepares to demonstrate the completed work to stakeholders. This is not about creating a polished presentation but about being ready to show the actual, working product.

You Don't Need a Gantt Chart You Need Clear Path

A visual approach to agile project management



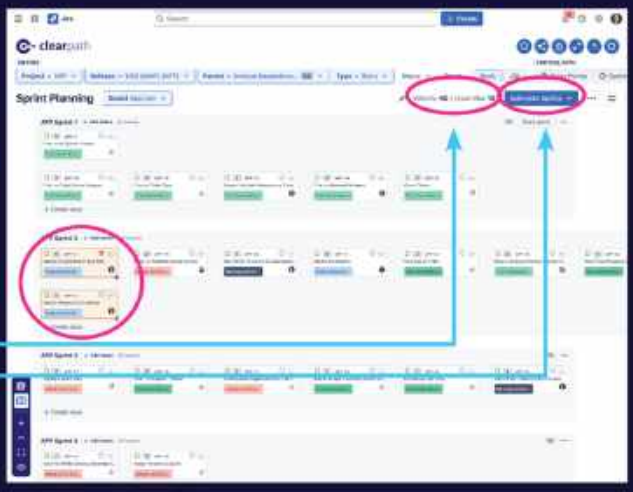
Story Points over hours. Projections over plans.

Less time in meetings, more time delivering value.

- **Plan Sprints with More Confidence**
- **Bring Certainty to the Execution Plan**
- **Bring Teams to a Shared Understanding Faster**
- **Manage the Backlog Effectively**
- **Simple Onboarding**
- **Built with Security**

Introducing Sprint Planning Mode!

- See potential risks with advanced warnings
- Customize Velocity and ticket quantity for results specific to your team
- Save time with auto plan sprints



clearpath by moserlabs

Contact us at atlassianapps@moserit.com

Download for free on Atlassian Marketplace

Visit our website to learn more about Clear Path

Daily Scrum Checklist

TIME	15 MINUTES	**CORDINATOR**	SCRUM MASTER
DATE	DAILY	**ATTENDEES**	SCRUM MASTER DEVELPERS
			PRODUCT OWNER

- [ ] Start on time
- [ ] Have Jira active sprint view ready
- [ ] Ensure team sticks to the scrum prompt
- [ ] STOP side conversations that get too much into the weeds
- [ ] Document blockers/impediments
- [ ] Listen for any impediments that are currently not in a blocked status in Jira
- [ ] Listen for any work in progress, but not reflected in Jira
- [ ] If someone is absent, encourage them to update via slack/email
- [ ] Be on the look out for stale work items in Jira
- [ ] Ask for any go backs before closing scrum out
- [ ] Schedule follow up calls for discussions that were cut short
- [ ] End meeting on time!

Sprint Review Checklist

TIME	60-90 MINUTES	**CORDINATOR**	SCRUM MASTER
DATE	LAST DAYOFSPRINT	**ATTENDEES**	ENTIRE TIEAM

1 PRE-REVIEW

- [ ] Check with team to determine items to be scheduled
- [ ] Define sprint review agenda
- [ ] Email out agenda to attendees
- [ ] For any developers not present, get recording of their demos
- [ ] Remind developers to prepare demo environments
- [ ] Make offering to demo gods

2 REVIEW

- [ ] Record the meeting!
- [ ] Start sprint review and summarize what attendees should expect to see
- [ ] Notify first presenter that they are next
- [ ] Go through each developer (team member)
- [ ] Ensure each developer does not exceed allocated demo time
- [ ] Take notes in Jira comments of action items made during the story demonstration
- [ ] Capture action items that need to be followed up on (non story related)
- [ ] Move PO approved stories to done
- [ ] Move any PO failed stories back to in progress
- [ ] Play video from any absent developers
- [ ] Make sure every developer demonstrates something
- [ ] Make closing comments, thank everyone for coming, and close review!

Sprint Retrospective Guide

Step 1: Set the Stage
The goal of this first step is to create a safe and positive environment where everyone feels comfortable sharing their honest thoughts.

- **Welcome Everyone:** Greet the team and briefly restate the purpose of the retrospective: to inspect the last Sprint and create a plan to improve.
- **Emphasize the Prime Directive:** Remind the team of the retrospective prime directive: "Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand."
- **Do a Quick Check-in:** Use a simple icebreaker or ask a question like, "In one word, how did the last Sprint feel?" to get everyone engaged.

Step 2: Gather Data
This step is about helping the team remember and share their experiences from the Sprint. The goal is to create a shared picture of what happened.

- **Choose an Activity:** Use a simple and engaging format to collect data. A classic method is "Mad, Sad, Glad":
  - Set up a Confluence retrospective board. Give everyone a few minutes to silently write down things from the Sprint that made them feel mad, sad, or glad.
- **Share and Cluster:** Have each person briefly share what they wrote as they place their sticky notes on a whiteboard. Group similar or related notes together to start identifying themes.

Step 3: Generate Insights
Now that you have the data, the goal is to dig deeper to find the root causes behind the key themes you've identified.

- **Identify Key Themes:** As a group, look at the clustered sticky notes and identify the most significant topics.
- **Facilitate a Discussion:** For the most important themes, facilitate a discussion. Ask powerful, open-ended questions like:
  - "Why do we think this happened?"
  - "What patterns do we see here?"
- **Use the "5 Whys" Technique:** For a particularly challenging issue, ask "Why?" five times to drill down from a symptom to its root cause.

Step 4: Decide What to Do
The most important step is to turn your insights into concrete, actionable improvements.

- **Brainstorm Solutions:** For the 1-2 most important issues identified, have the team brainstorm potential solutions or experiments to try in the next Sprint.
- **Create Action Items:** Collaboratively create 1-3 specific, measurable, achievable, relevant, and time-bound (SMART) action items.
- **Assign Owners:** Each action item should have a volunteer from the team who will be responsible for seeing it through.

Step 5: Close the Retrospective
End the meeting on a positive and forward-looking note.

- **Summarize Action Items:** Quickly recap the action items and their owners to ensure everyone is clear on the plan.
- **Appreciate and Thank:** Thank everyone for their active participation and honest feedback.
- **Do a Quick Check-out:** You can do a quick "fist of five" for confidence in the next Sprint or ask for a closing word from each person. This provides a sense of closure.

Retrospective Formats Library

Start / Stop / Continue

- Ask the team: What should we start doing? What should we stop doing? What should we continue doing?
- Good for simple reflection and quick action items.

Mad / Sad / Glad

- Team members share what frustrated them (mad), what disappointed them (sad), and what made them happy (glad).
- Great for uncovering emotions and morale issues.

4Ls (Liked, Learned, Lacked, Longed For)

- Encourages balanced reflection: things enjoyed, lessons learned, gaps noticed, and wishes for improvement.

Sailboat (a.k.a. Speedboat)

- Visual metaphor:
  - Wind = things helping the team
  - Anchors = things holding the team back
  - Rocks = risks ahead
  - Island = goals
- Fun, visual, and metaphor-driven.

Keep / Problem / Try (KPT)

- Simple categorization: what's working, what's problematic, and what to try next.
- Helps drive experimentation.

Hot Air Balloon

- Balloon = things lifting us up
- Sandbags = things weighing us down
- Storms = external risks/challenges
- Sun = positive forces ahead

Starfish

- Categories: Start, Stop, More Of, Less Of, Keep Doing.
- Provides more granularity than Start/Stop/Continue.

Timeline Retrospective

- Map key events of the sprint/PI along a timeline.
- Encourages storytelling and uncovering hidden frustrations.

Weather Report

- People rate the sprint using weather icons ( sunny, cloudy,  rainy,  stormy).
- Quick pulse check of morale.

Movie Critic

- Rate the sprint like a movie: best scenes, worst scenes, plot twists, cliffhangers.
- Playful and engaging for creative teams.

5 Whys

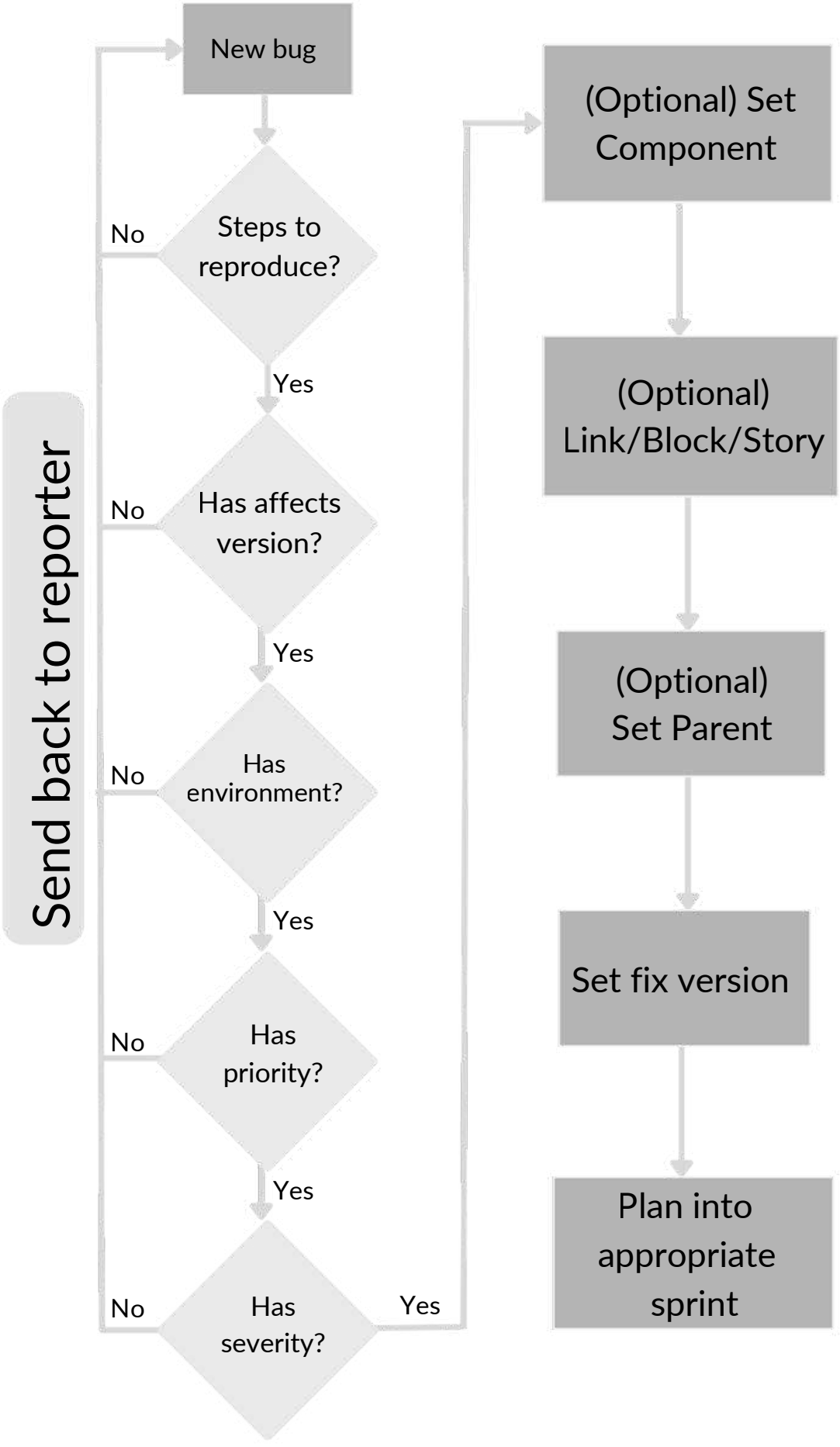
- Pick one issue and ask "why?" five times to get to the root cause.
- Good for teams who struggle with recurring problems.

Sprint Retrospective Checklist

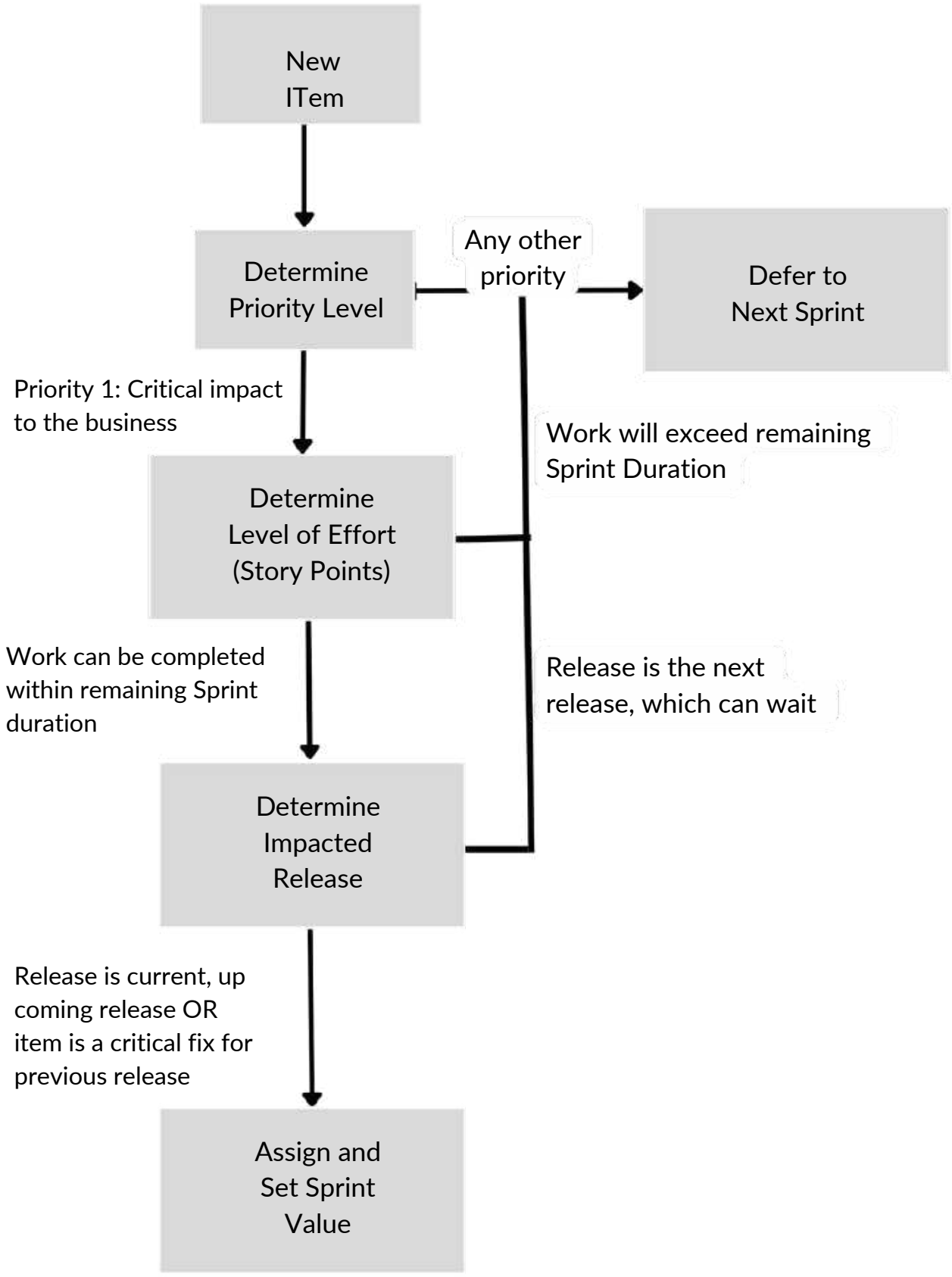
TIME	60-120 MINUTES	**CORDINATOR**	SCRUM MASTER
DATE	LAST DAY OF SPRINT	**ATTENDEES**	SCRUM MASTER & DEVELOPERS ONLY

- [ ] Make sure only Scrum Master & developers are present
- [ ] Start with an ice breaker
- [ ] Encourage people to bring lunch or snack
- [ ] Perform a trust building exercise
- [ ] Use Confluence to take notes
- [ ] If in-person, give everyone sticky notes to answer prompt
- [ ] If virtual, ensure everyone makes a comment on each prompt
- [ ] Take attendance in Confluence so members get a link to notes
- [ ] Capture any actions items discussed
- [ ] Follow up on each action item before next restrospective
- [ ] Address challenges/friction within the team
- [ ] Focus on process/people feedback, not technical challenges

Triaging Bugs Guide



How To Deal With Scope Creep Guide



Product Owner Playbook

Step 1: Laying the Foundation - Define the "Why" and "Who"
Before a single feature is built, the Product Owner must establish a clear vision and understand the audience. This foundation guides all future development decisions.

- **Create the Product Vision**: Work with key stakeholders to define the overarching goal and purpose of the product. Use a tool like a Product Vision Board to articulate the target users, their needs, the key features, and the business goals.
- **Identify and Analyze Stakeholders:** Create a list of all stakeholders (customers, users, internal departments, executives). Use a Stakeholder Map to understand their level of influence and interest, which will inform your communication strategy.
- **Develop User Personas:** Create detailed, fictional profiles of your key user segments. This helps the entire Scrum Team empathize with users and make user-centric decisions.

Step 2: Building and Prioritizing the Product Backlog
With a clear vision, the next step is to translate that vision into a tangible, ordered list of work. The Product Backlog is the single source of truth for what the team will build.

- **Gather Requirements:** Use techniques like user story mapping, interviews, and competitor analysis to gather potential features and requirements.
- **Write Effective User Stories**: Frame the work from a user's perspective. Each Product Backlog Item (PBI) should be understandable, estimable, and testable.
- **Prioritize Ruthlessly:** The most crucial job of a PO is to order the backlog to maximize value. Use a clear technique to make decisions:
  - Value vs. Effort Matrix: Plot items to find high-value, low-effort "quick wins."
  - MoSCoW Method: Classify features as Must-have, Should-have, Could-have, or Won't-have for a specific release.
- **Conduct Backlog Refinement:** Regularly meet with the Developers to add details and estimates to upcoming PBIs, ensuring they are ready for future Sprints.

Step 3: Executing and Guiding the Sprint
During the Sprint, the Product Owner's role shifts to providing clarity and making tactical decisions to ensure the Sprint Goal is met.

- **During Sprint Planning:**
  - Present the highest-ordered Product Backlog items.
  - Propose a Sprint Goal that provides a cohesive objective for the Sprint.
  - Be available to answer the Developers' questions so they can select the work and create their plan.
- **Throughout the Sprint:**
  - Be accessible to the Developers to clarify requirements.
  - Accept completed stories as they meet the Definition of Done, providing quick feedback.
- **During the Sprint Review:**
  - Present the completed Increment to stakeholders.
  - Facilitate a discussion to gather valuable feedback.
  - Use this feedback to make adjustments to the Product Backlog.

Step 4: Closing the Loop - Feedback and Adaptation
The work of a Product Owner is never truly done. The final step is a continuous cycle of releasing value, gathering feedback, and adapting the plan.

- **Engage with Users:** Actively seek feedback on the released Increment through user interviews, surveys, and analytics.
- **Communicate with Stakeholders:** Keep stakeholders informed of progress and any changes to the product roadmap. Practice transparency about what is and isn't being worked on.
- **Adapt the Product Backlog:** Use the feedback and data gathered to make informed decisions about what to build next. The Product Backlog is a living artifact that should evolve based on what is learned.

Project Manager Playbook

Here is a step-by-step playbook for a Project Manager, structured around the traditional phases of project management. This guide is ideal for managing projects that require a more formal structure and planning upfront.

Step 1: Initiation Phase - Defining the Project
This initial phase is about defining the project at a high level and securing the necessary authority to proceed. The goal is to ensure the project is aligned with business objectives before significant resources are committed.

- **Develop a Project Charter:** This is the project's foundational document. It should include the business case, high-level goals, scope, key stakeholders, and the project manager's authority. Its approval formally authorizes the project.
- **Identify Stakeholders:** Create a stakeholder register to list everyone who is impacted by or has an interest in the project. Analyze their influence and expectations to create a communication plan later.
- **Conduct a Feasibility Study:** Assess the project's viability in terms of budget, timeline, and resources to confirm it's a worthwhile investment.

Step 2: Planning Phase - Creating the Roadmap
This is the most intensive phase, where you create a detailed plan that will guide the team through execution and control. A thorough plan is the key to managing scope, time, and budget.

- **Define Scope:** Create a detailed scope statement that clearly outlines what is and is not included in the project.
- **Create a Work Breakdown Structure (WBS):** Break down the major project deliverables into smaller, more manageable work packages. The WBS is the foundation for all other planning.
- **Develop the Schedule:** Use the WBS to sequence activities, estimate their duration, and create a project schedule. A Gantt chart is the most common tool for visualizing this timeline.
- Plan Resources, Budget, and Risks:
    - **Resource Plan:** Define the roles, responsibilities, and team structure.
    - **Budget:** Create a detailed cost estimate for all labor, materials, and other expenses.
    - **Risk Management Plan:** Identify potential risks, analyze their impact, and plan response strategies.
- Leverage Jira Plans (if you are a Jira Premium/Enterprise customer).
    - Jira Plans will help you define all of this and help you organize your project within Jira in a more “traditional” way as opposed to only relying on backlog/board.

Step 3: Execution Phase - Getting the Work Done
In this phase, the project plan is put into action. The project manager's focus shifts from planning to leading the team and managing communication.

- **Lead and Manage the Team:** Assign tasks, provide direction, and motivate the project team to perform their work.
- **Manage Communications:** Execute the communication plan to keep all stakeholders informed. This includes running status meetings, distributing reports, and ensuring a steady flow of information.
- **Engage Stakeholders:** Actively manage stakeholder expectations and ensure they remain engaged and supportive throughout the project.

Step 4: Monitoring & Controlling Phase - Tracking Progress
This phase runs in parallel with the Execution phase. It's about measuring project performance against the plan and taking corrective action when necessary.

- **Track Performance:** Use Key Performance Indicators (KPIs) to monitor progress against the schedule, budget, and scope baselines.
- **Manage Changes:** Implement a formal change control process. Any requested changes to the project scope, schedule, or budget must be formally reviewed, approved, and documented.
- **Monitor Risks:** Continuously keep an eye on identified risks and look for any new ones that may arise. Implement risk response plans as needed.

Step 5: Closing Phase - Finishing the Project
The final phase involves formally closing the project and ensuring all work is completed to the satisfaction of the stakeholders.

- **Obtain Formal Acceptance:** Deliver the final product or service and get formal sign-off from the client or sponsor that the project deliverables meet the agreed-upon requirements.
- **Conduct a Lessons Learned Session:** Hold a meeting with the project team to discuss what went well, what didn't, and what could be improved for future projects.
- **Archive Project Documents:** Organize and store all project documentation in a central repository for future reference.
- **Release the Team:** Formally release the project team members, ensuring their contributions are recognized.

Product Manager Playbook

Step 1: Discovery and Research - Find the Problem
Before building anything, your primary job is to become an expert on the market, the user, and the problem you're trying to solve.

- **Identify Market Problems:** Talk to potential and existing customers, sales teams, and support staff to uncover unmet needs and pain points.
- **Conduct User Research:** Go beyond what users say and observe what they do. Use techniques like user interviews, surveys, and usability tests to develop deep empathy for their experience.
- **Analyze the Competition:** Perform a thorough competitive analysis to understand the market landscape, identify gaps, and determine how your product can uniquely win.

Step 2: Strategy and Vision - Chart the Course
Once you understand the problem space, you must define a clear and compelling vision for your product and create a high-level plan to get there.

- **Define the Product Vision:** Craft a concise statement that describes the future state you are trying to create. This vision becomes the North Star for your entire team.
- **Set Goals and Objectives:** Use a framework like Objectives and Key Results (OKRs) to set measurable, ambitious goals for the product. This aligns the team around specific outcomes.
- **Create the Product Roadmap:** Develop a high-level, strategic roadmap that visualizes the direction of the product over the next several quarters. This is a communication tool for stakeholders that focuses on outcomes, not just a list of features.

Step 3: Planning and Prioritization - Decide What to Build
With a strategy in place, you must translate it into an actionable plan. This involves making tough decisions about what to build next to deliver the most value.

- **Develop a Business Case:** For significant initiatives, justify the investment by outlining the expected costs, revenue, and strategic impact.
- **Create a Prioritization Framework:** Use a structured method to remove bias from your decision-making. Common frameworks include:
  - **RICE**: (Reach, Impact, Confidence, Effort) - A quantitative model for scoring features.
  - **Kano Model:** Categorizes features based on their ability to satisfy customers (e.g., Basic, Performance, Excitement).
- **Write Product Requirements:** Create a Product Requirements Document (PRD) or use Agile epics and user stories to clearly define the problem, user needs, and acceptance criteria for the features you plan to build.

Step 4: Execution and Go-to-Market - Ship It
During this phase, you work closely with engineering to get the product built and with marketing and sales to launch it successfully.

- **Collaborate with Development:** Act as the voice of the customer during the development process. Participate in Agile ceremonies like sprint planning and sprint reviews to ensure the final product meets the requirements.
- **Develop a Go-to-Market (GTM) Plan:** Work with marketing, sales, and support to create a comprehensive launch plan. This includes positioning, messaging, pricing, and channel strategy.
- **Enable the Sales and Support Teams:** Provide your internal teams with the training and documentation they need to effectively sell and support the new product or feature.

Step 5: Growth and Iteration - Measure and Adapt
Your job isn't done at launch. The final step is a continuous loop of measuring performance, learning from users, and iterating on the product.

- **Analyze Product Metrics:** Define and track Key Performance Indicators (KPIs) related to user engagement, retention, and satisfaction. Use this data to understand what's working and what isn't.
- **Gather User Feedback:** Create channels for continuous feedback, such as in-app surveys, customer advisory boards, and feature request systems.
- **Manage the Product Lifecycle:** Use the data and feedback you've gathered to make decisions about what to improve, what new features to build, and when to retire old features or products.

THE CORE OF YOUR SYSTEM OF WORK

Build smarter with Jira and Microsoft 365

Microsoft 365 for Jira

Explore use cases for all your teams and integrate Jira seamlessly with Microsoft Teams, Outlook, and more. Boost collaboration, streamline support, and empower your teams to deliver faster.

MICROSOFT 365 FOR JIRA – KEY FEATURES

MICROSOFT TEAMS:
Use Microsoft Teams to work on Jira and JSM work items. Start chats from Jira and manage requests via the JSM portal in Teams.

OUTLOOK:
Turn emails into Jira work items and send emails directly from Jira.

CALENDAR:
Sync Jira issues with Outlook calendar and Microsoft Bookings for seamless scheduling.

TO DO:
Manage personal tasks seamlessly connected to Jira.

TRY IT FOR FREE!
Atlassian Marketplace
yasoon.com

Platinum Marketplace Partner

ISO 27001 | SOC 2 certified

YASOON

Technical Lead Playbook

Step 1: Establish Technical Vision and Architecture
Your first responsibility is to define the technical foundation and long-term strategy. This ensures the team is building a robust and scalable system.

- **Define the Technical Strategy:** Collaborate with product managers and architects to create a technical roadmap that aligns with the product vision.
- **Choose the Right Tools**: Lead the evaluation and selection of technologies, frameworks, and platforms. Focus on creating a tech stack that is fit for purpose and maintainable.
- **Set Architectural Standards:** Document and communicate clear architectural principles and design patterns for the team to follow. This includes standards for security, performance, and reliability.

Step 2: Guide the Day-to-Day Development Process
You are the hands-on leader who ensures the team is building things the right way. This involves active participation in the entire development cycle.

- **Lead Technical Design Sessions:** Facilitate sessions where the team breaks down complex features into manageable technical tasks. Guide discussions to ensure the proposed solution is sound.
- **Champion Code Quality:** Set the standard for code quality. Lead by example in your own code and establish a rigorous but constructive code review process. Ensure meaningful feedback is given and received.
- **Implement a Testing Strategy:** Work with the team to define and implement a comprehensive testing strategy, including unit, integration, and end-to-end tests, to ensure software quality.

Step 3: Mentor and Grow the Team
A key part of your role is to elevate the technical skills of every engineer on your team. You are a multiplier of the team's talent.

- **Mentor and Coach:** Provide regular one-on-one mentorship to developers, especially junior and mid-level engineers. Help them with their career growth and technical challenges.
- **Unblock Team Members:** Be the first point of contact when a developer is stuck on a difficult technical problem. Help them think through the issue and guide them toward a solution.
- **Foster Knowledge Sharing:** Create a culture of learning by encouraging tech talks, pair programming sessions, and the documentation of complex systems.

Step 4: Manage Technical Debt and Operations
Your responsibilities extend beyond shipping new features. You must also ensure the long-term health and stability of the system.

- **Manage Technical Debt:** Create a strategy for identifying, prioritizing, and paying down technical debt. Ensure there is a healthy balance between building new features and improving existing code.
- **Ensure Smooth Deployments:** Own the team's CI/CD (Continuous Integration/Continuous Deployment) pipeline. Work to make deployments automated, safe, and frequent.
- **Lead Incident Response:** When production issues occur, you are the technical leader for the incident response. You guide the team in diagnosing the problem, implementing a fix, and conducting a post-mortem to prevent future occurrences.

QA Playbook

Step 1: Test Planning and Strategy

Before any testing begins, you need a clear plan. This phase is about defining the scope, objectives, and approach to ensure everyone understands what quality means for the project.

- Understand Requirements: Work closely with product managers and developers to thoroughly understand the features, user stories, and acceptance criteria.
- Define Test Scope: Clearly identify what will be tested (in-scope) and what will not be tested (out-of-scope).
- Choose Testing Types: Determine the right mix of testing for the project, such as:
  - Manual Testing: For exploratory and usability testing.
  - Automated Testing: For repetitive regression tests.
  - Performance Testing: To check speed and stability under load.
- Set Up the Test Environment: Ensure a stable and dedicated environment is available for testing, with the necessary hardware, software, and data.

Step 2: Test Design and Development

With a plan in place, the next step is to create the specific assets you will use for testing. Clear and well-designed tests are essential for effective quality control.

- Write Test Cases: Create detailed, step-by-step test cases for each feature. A good test case includes a clear description, steps for execution, expected results, and actual results.
- Develop Automation Scripts: For the parts of the application covered by automation, write robust and maintainable test scripts.
- Prepare Test Data: Create or acquire the data needed to run your test cases. This might include creating sample user accounts or generating specific data sets to test edge cases.

Step 3: Test Execution and Defect Management

This is the active phase where you run your tests and find bugs. The goal is to execute the plan and clearly communicate any issues you discover.

- Execute Tests: Run the manual test cases and automated scripts according to the test plan.
- Log Defects: When a test fails, log a bug in your tracking system (like Jira). A high-quality bug report includes:
  - A clear and concise title.
  - Steps to reproduce the bug.
  - The expected result vs. the actual result.
  - Screenshots or video recordings.
- Triage and Track Bugs: Work with the product manager and developers to prioritize bugs. Track the bug through its entire lifecycle, from being reported to being fixed and verified.

Step 4: Reporting and Continuous Improvement

The final step is to communicate the state of the product's quality and learn from the process to make it better next time.

- Create Test Summary Reports: At the end of a testing cycle, create a report that summarizes the testing activities. This should include metrics like the number of test cases executed, pass/fail rates, and the number of open vs. closed bugs.
- Analyze Results: Look for trends in the defects. Are bugs concentrated in a specific part of the application? This can help identify areas that need more attention.
- Conduct a Retrospective: Hold a meeting to discuss what went well in the QA process and what could be improved. Use this feedback to refine your playbook for the next project.

Developer Playbook

Step 1: Understand the Work
Before writing a single line of code, your goal is to have complete clarity on the "what" and the "why." A clear understanding upfront prevents rework and ensures you build the right solution.

- **Participate in Planning:** Actively engage in sprint planning or backlog refinement sessions. Ask clarifying questions to understand the business value and user impact of a feature.
- **Deconstruct the Task:** Break down the user story or task into smaller, manageable sub-tasks. This helps you create a clear plan of action and identify potential complexities early.
- **Collaborate on a Technical Plan:** For complex features, work with your Tech Lead or other senior engineers to create a high-level technical design. This ensures your approach aligns with the team's architectural standards.

Step 2: The Development Loop
This is the core "build" phase. The focus is on writing clean, maintainable code and maintaining a tight feedback loop through frequent commits and communication.

- **Write Clean Code:** Follow your team's established coding standards. Write code that is simple, readable, and easy for other developers to understand and maintain.
- **Commit Frequently:** Make small, logical commits to your version control system (like Git). Write clear and descriptive commit messages that explain the purpose of the change. This creates a clean history and makes code reviews easier.
- **Open a Pull Request (PR) Early:** Don't wait until you are completely finished to open a PR. Pushing your code early (even as a "Work in Progress") allows teammates to see your approach and provide feedback before you've gone too far down a path.

S**tep 3: Ensure Quality and Collaborate**
You are responsible for the quality of your own code. This step is about verifying your work and collaborating with your peers to ensure the entire team is shipping a high-quality product.

- **Write Your Tests:** Write unit and integration tests to cover your code. Good tests act as living documentation and prevent future regressions.
- **Conduct Thorough Code Reviews:** When reviewing a teammate's PR, provide constructive, respectful, and actionable feedback. When receiving feedback, be open and receptive. The goal is to improve the code, not to criticize the person.
- **Work with QA:** Collaborate closely with the Quality Assurance team. Be responsive to the bugs they file and provide them with the information they need to test your features effectively.

Step 4: Ship It and Learn
Getting your code into production is the goal, but your job doesn't end there. This final step is about ensuring your feature is working as expected and learning from the process.

- **Merge and Deploy:** Once your PR is approved and has passed all automated checks, merge it into the main branch. Support the deployment process and be available to help if any issues arise.
- **Monitor Your Feature:** After deployment, keep an eye on monitoring and logging dashboards to ensure your feature is performing as expected in production.
- **Participate in the Retrospective:** Actively participate in the team's retrospective. Share what went well, what challenges you faced, and how the team can improve its process in the next cycle.

Story Points vs. Time Estimates Cheat Sheet

Aspect	Story Points	Time Estimates
What it Measure	A relative measure of effort, complexity, and uncertainty. It answers, "How big is this compared to that?"	An absolute measure of duration. It answers, "How long will this take in hours or days?"
Type of Scale	Relative (e.g., Fibonacci sequence: 1, 2, 3, 5, 8...). The numbers only have meaning in relation to each other.	Absolute (e.g., hours, days). Each number has a fixed, real-world meaning.
Precision	Low precision (abstract). Intentionally abstract to avoid false precision and focus on the overall size.	High precision (concrete). Tries to be an exact prediction of the time required.
Impact of Team Changes	Less impacted. Since the points are relative to the team's own baseline, the team's velocity will naturally adjust over time.	Highly impacted. A task estimated at 8 hours for a senior developer might take a junior developer 16 hours.
Focus	Encourages a team conversation about complexity and effort.	Encourages a conversation about duration and deadlines.
Best For	Long-term forecasting (velocity) and prioritizing the product backlog based on value vs. effort.	Short-term, detailed planning where a specific timeline is required for a small batch of work.

Key Takeaway

- Use **Story Points** to understand the **size** of the work.
- Use **Time Estimates** to predict the **duration** of the work.

Definition of Ready &
Definition of Done Examples

Definition of Ready (DoR)
The **Definition of Ready** is a checklist of criteria that a user story must meet **before** it can be pulled into a Sprint. It's an agreement between the Product Owner and the Developers to ensure that stories are well-understood and immediately actionable.

Purpose: To prevent half-baked ideas from entering a Sprint, which reduces ambiguity and ensures a smoother workflow.

Example Checklist:
[ ] The user story is written from a user's perspective.
[ ] Acceptance criteria are clear and testable.
[ ] The story has been estimated by the Developers.
[ ] Any dependencies on other teams or stories have been identified.
[ ] The necessary designs or mockups are attached and approved.

Definition of Done (DoD)
The **Definition of Done** is a checklist of all the activities that must be completed for a user story **before** it can be considered "done" and part of the product Increment. It is a shared understanding of what it means for work to be complete.

Purpose: To ensure a high level of quality and consistency for all work the team delivers.

Example Checklist:
[ ] The code is written and peer-reviewed.
[ ] All unit and integration tests are passing.
[ ] The feature meets all acceptance criteria.
[ ] The code has been merged into the main branch.
[ ] The feature has been tested and verified by QA.

Key Difference

- **Definition of Ready** is the entry criteria for a Sprint.
- **Definition of Done** is the exit criteria for a Sprint.

Table Filter, Charts & Spreadsheets
for Confluence

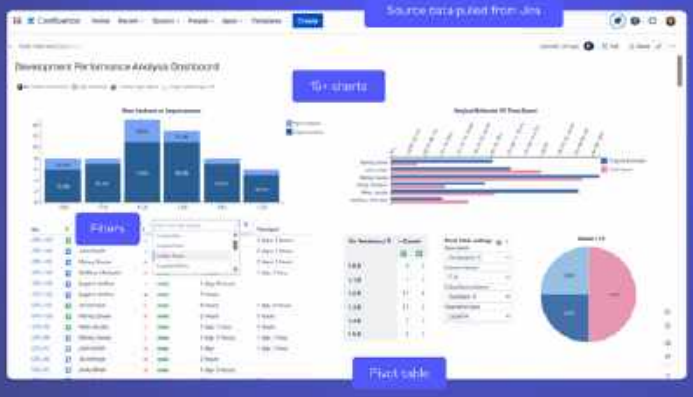
Platinum
Marketplace Partner

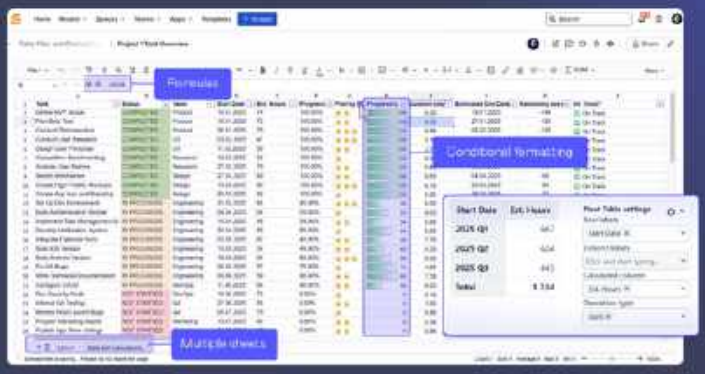
STILTSOFT

Make your data LIVE in Confluence with the most comprehensive reporting solution!

Show your impact with always-on reports

Turn your data into stories with advanced filter and charts, connect live input, and reuse content



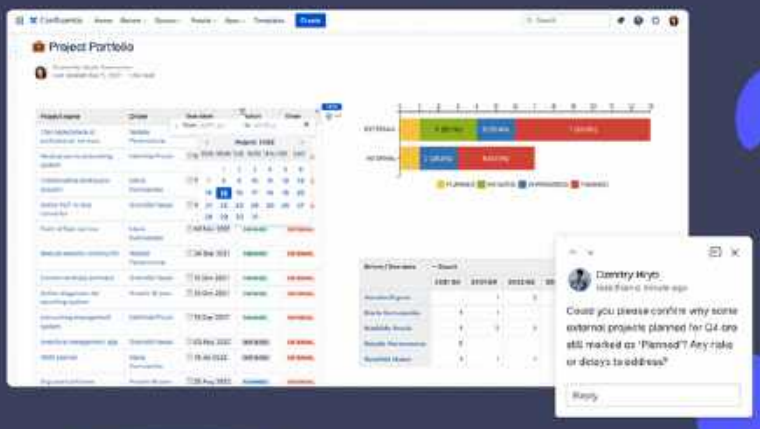


Give your tables the power of Excel

Create pivot tables, calculate formulas in spreadsheets,and merge tables with SQL

Make Confluence a single source of truth

Collaborate on data, make decisions, and take action to get everyone on the same page



★★★★

Very powerful add-on. Essential for any serious dashboarding and data wrangling.

Stathis Kanterakis, Principal Scientist

PI Planning Runbook

A **Program Increment (PI) Planning Runbook** is a step-by-step guide for facilitating the PI Planning event, a cornerstone of the Scaled Agile Framework (SAFe). The event's purpose is to align all teams on an Agile Release Train (ART) to a shared mission and vision for the next 8-12 weeks.

Phase 1: Preparation (Before the Event)
Success in PI Planning is heavily dependent on preparation. The goal is to have the vision and top priorities ready for the teams.

- **Define Business Context:** Leadership prepares a presentation on the current state of the business and the vision for the future.
- **Prepare the Top 10 Features:** Product Management refines and prioritizes the top 10 features for the upcoming Program Increment.
- **Present Architectural Vision:** The System Architect or senior technical leaders prepare a briefing on the architectural runway and any technical guidelines.
- **Logistics:** The event facilitator (often a Release Train Engineer or RTE) ensures all logistics are handled, whether the event is in-person, remote, or hybrid.

Phase 2: The PI Planning Event (2 Days)

Day 1: Planning

- **Business Context (Agenda Item 1):** A senior executive presents the business vision and goals for the upcoming PI.
- **Product/Solution Vision (Agenda Item 2)**: Product Management presents the top features from the program backlog.
- **Architectural Vision (Agenda Item 3):** The architect presents the technology vision, new enablers, and any system-level changes.
- **Team Breakouts #1 (Agenda Item 4):** Teams meet to create their draft plans for the upcoming iterations. They identify risks, write draft PI Objectives, and map out dependencies on other teams using a program board.
- **Draft Plan Review (Agenda Item 5):** Each team presents a summary of their draft plan, highlighting their capacity, draft objectives, and key risks.

Day 2: Finalizing the Plan

- **Planning Adjustments (Agenda Item 6):** Based on feedback from the draft plan review, teams adjust their plans and resolve dependencies with other teams.
- **Team Breakouts #2 (Agenda Item 7):** Teams finalize their iteration plans and PI Objectives. Business Owners assign a business value (on a scale of 1-10) to each objective.
- **Final Plan Review (Agenda Item 8):** Each team presents its final plan and PI Objectives to the entire group for review.

- **Address Program Risks (Agenda Item 9):** All identified risks are discussed and categorized using the ROAM method:
  - Resolved: The risk is no longer a concern.
  - Owned: Someone takes ownership of managing the risk.
  - Accepted: The risk is understood and accepted as is.
  - Mitigated: A plan is created to reduce the impact of the risk.
- **Confidence Vote (Agenda Item 10):** The entire Agile Release Train votes on their confidence in meeting the PI Objectives. This is typically done on a scale of 1 to 5. If the average is low, the plan may need to be reworked.
- **Plan Rework (If Necessary):** If confidence is low, the team reworks the plan until a sufficient level of confidence is achieved.

Phase 3: Post-Event Follow-Up
The work isn't done when the meeting ends. The outputs of PI Planning must be captured and used to guide the execution of the Program Increment.

- **Synthesize Objectives:** The Release Train Engineer (RTE) collects and summarizes the PI Objectives from all teams into a program-level view.
- **Input into Tooling:** All stories, features, and dependencies are loaded into the team's Agile project management tool (e.g., Jira Align, Rally).
- **Schedule Follow-ups:** Any necessary follow-up meetings for owned or mitigated risks are scheduled.

Capacity Planning Worksheet

Purpose: To determine the team's total available work capacity for a Sprint. This helps the team create a realistic and achievable Sprint Backlog during Sprint Planning.

Step 1: Sprint Details

- **Sprint Name/ Number:** *Sprint 25-20*
- **Start Date:** *September 29, 2025*
- **End Date:** *October 10, 2025*
- **Total Business Days in Sprint:** *10*

Step 2: Team Capacity Calculation

Fill out the table below for each member of the team.

Team Member Name	Total Business Days in Sprint	Days Off (Vacation, Holidays)	Available Days	Productive Hours per Day *	Total Available Hours
Example: Alice	10	1	9	6	54
Example: Bob	10	0	10	6	60
Example: Charlie	10	3	7	6	42
TOTAL					**156**

Note on Productive Hours per Day: A standard 8-hour day does not equal 8 hours of project work. Account for meetings, emails, and other daily activities. A realistic number is often between 5-6 hours. Your team should agree on this number and adjust it based on experience.

Step 3: Using Your Capacity

Now that you have your total capacity number, here's how to use it in Sprint Planning:

For Teams Using Time Estimates (Hours):

- **The Total Available Hours** (e.g., **156 hours** in the example) is the maximum number of hours the team should commit to in the Sprint Backlog.
- As the team selects work, sum the time estimates for each task. The total should not exceed the team's capacity.

For Teams Using Story Points:

- Your primary guide for how much work to pull into a Sprint is your team's historical velocity (the average number of story points completed in previous Sprints).
  - The capacity worksheet is still a valuable tool to understand why your velocity might change for a given Sprint.
- **Example:** If your average velocity is **25 story points**, but the capacity worksheet shows that two team members are on vacation, you know you should commit to fewer than 25 points. This calculation provides the data to support that decision.

By taking a few minutes to calculate capacity, your team can create a much more realistic and achievable plan, leading to more successful and less stressful Sprints.

Release Planning Guide

Step 1: Preparation (The Inputs)
Before the planning session, you need to gather the necessary inputs to ensure the meeting is productive.

- **Prioritized Product Backlog:** The Product Owner should have a well-ordered backlog with the most important features at the top.
- **Estimated Stories:** The development team should have provided relative estimates (like story points) for a significant portion of the backlog items.
- **Team Velocity:** You need the team's average velocity from the last 3-5 Sprints. This is the key metric used for forecasting.
- **Product Vision:** A clear vision and goal for the release, so the team understands the "why" behind the plan.

Step 2: The Release Planning Event
This is a collaborative meeting involving the Product Owner, the development team, and the Scrum Master.

- **Present the Vision:** The Product Owner kicks off the meeting by presenting the release goal and the key features to be delivered.
- **Review Capacity and Velocity:** The team reviews their average velocity and notes any known capacity changes for the upcoming Sprints (e.g., holidays, team members on vacation).
- **Map Features to Sprints:** The team maps out the Sprints in the release and starts allocating features from the top of the backlog to each Sprint based on their velocity.
    - **Example:** If a team's velocity is 25 story points and the release will be over 4 Sprints, they can forecast completing approximately 100 points of work. They would pull the highest-priority stories from the backlog, up to a total of 100 points, and lay them out across the Sprints.
- **Identify Risks and Dependencies:** The team discusses potential risks that could impact the release and identifies any dependencies on other teams.

Step 3: Executing and Adapting the Plan
An Agile release plan is a forecast, not a guarantee. It is meant to be adapted as the team learns more.

- **Communicate the Plan:** The release plan is shared with stakeholders to provide visibility into the upcoming work. It's crucial to communicate that this is a forecast and is subject to change.
- **Track Progress:** At the end of each Sprint during the Sprint Review, the team should review their progress against the release plan.
- **Adapt as Needed:** Based on feedback and the team's actual velocity, the Product Owner may need to adjust the release plan by re-prioritizing or removing features. This ensures the plan always reflects the current reality.

Agile Reporting in Jira

Report	Board Type	Purpose	How to Use It
Burndown Chart	Scrum	Tracks remaining work in a sprint	Use it to see if the team is on track to complete the committed sprint work. A steep drop means completed work; a flat line means no progress.
Burnup Chart	Scrum	Shows work completed vs. total scope	Use it to visualize both progress and scope changes (scope creep is obvious if the top line rises).
Sprint Report	Scrum	Summarizes completed and incomplete sprint work	Review at sprint retrospectives to discuss carry-over work and blockers.
Velocity Chart	Scrum	Shows work completed per sprint	Use it for forecasting and capacity planning. Helps teams understand their sustainable pace.
Epic Report	Scrum & Kanban	Tracks progress of an epic over time	Use it to see completed vs. remaining work in an epic, and how quickly progress is being made.
Epic Burndown	Scrum	Shows how an epic is burning down across sprints	Great for long-term epic tracking and spotting scope creep within epics.
Release Burndown	Scrum	Tracks release progress across multiple sprints	Use it to monitor if the team will hit release goals and to adjust scope or resources if needed.
Cumulative Flow Diagram (CFD)	Kanban & Scrum	Visualizes work in different workflow states	Use it to detect bottlenecks (e.g., if “In Progress” keeps expanding) and ensure a smooth flow.

Report	Board Type	Purpose	How to Use It
Control Chart	Kanban & Scrum	Measures cycle time & lead time	Use it to assess predictability. A stable chart means consistent delivery; spikes mean bottlenecks.
Version Report	Scrum	Tracks progress toward a version (release)	Use it to forecast whether a release will be delivered on time. Considers velocity and scope.
Created vs. Resolved work items	Kanban & Scrum	Compares work items created vs. resolved over time	Use it to see if the team is keeping pace with incoming work or accumulating backlog.
Average Age Report	Kanban & Scrum	Shows average age of unresolved work items	Use it to detect lingering work items that need attention.
Pie Chart Report	Kanban & Scrum	Visualizes work items by a chosen field (e.g., status, assignee, priority)	Use it for high-level breakdowns of work distribution.
User Workload Report	Kanban & Scrum	Shows number of work items assigned per user	Use it to spot uneven workload distribution across team members.
Time Tracking Report	Kanban & Scrum	Compares estimated vs. actual time spent on work items	Use it to improve estimation accuracy and track effort.
Resolution Time Report	Kanban & Scrum	Shows average time to resolve work items	Use it to track efficiency and help with SLA-type commitments.

Scaling Agile Frameworks Overview

1. Scaled Agile Framework (SAFe®)
SAFe is a highly structured and prescriptive framework designed for large-scale enterprise agility. It organizes multiple Agile teams into a virtual team-of-teams called an **Agile Release Train (ART)**, which works together to deliver value.

- **Core Philosophy:** To provide alignment, collaboration, and delivery for a large number of Agile teams. It is a top-down approach that provides a lot of guidance and defined roles.
- **Key Event:** Program Increment (PI) Planning, a multi-day, face-to-face event where all teams on the ART plan their work together for the next 8-12 weeks.
- **Best For:** Large, hierarchical organizations that need a high degree of predictability and a structured, prescriptive approach to scaling.

2. Large-Scale Scrum (LeSS)
LeSS is a framework for scaling Scrum that emphasizes simplicity. It applies the principles and roles of a single Scrum team to multiple teams working on the same product.

- **Core Philosophy:** "More with LeSS." It aims to scale Scrum with minimal additional process, roles, or artifacts. It is a bottom-up approach that centers around having one Product Owner and one Product Backlog for all teams.
- **Structure:** LeSS has two variants: LeSS for 2-8 teams and LeSS Huge for more than 8 teams.
- **Best For:** Organizations that are already proficient in Scrum and want to scale without adding significant organizational overhead.

3. Nexus™
Nexus is a lightweight framework from Scrum.org (the organization of Scrum co-creator Ken Schwaber). It is designed to guide 3-9 Scrum teams in delivering an integrated increment of a product each Sprint.

- **Core Philosophy:** To provide an "exoskeleton" for Scrum. It adds a new role (the Nexus Integration Team) and new events (like the Nexus Daily Scrum) to manage dependencies and integration issues between the teams.
- **Focus:** Its primary focus is on resolving the integration challenges that arise when multiple teams work from a single Product Backlog.
- **Best For:** Organizations that want a simple extension to their existing Scrum practice to manage the work of several teams on one product.

4. Disciplined Agile (DA)
Disciplined Agile is a flexible and adaptable toolkit acquired by the Project Management Institute (PMI). It's not a single framework but a collection of strategies and practices from various sources like Scrum, Kanban, and Lean.

- **Core Philosophy:** "Choose your Way of Working (WoW)." It is a goal-driven approach that provides guidance to help organizations select the best practices for their specific context.
- **Structure:** It is a decision-making framework that helps you "stitch together" a process that is right for your team.
- **Best For:** Organizations that prefer a flexible, adaptable toolkit over a single, prescribed framework.

Agile Anti-Patterns List

An **Agile Anti-Pattern** is a common practice that may seem beneficial on the surface but ultimately undermines the principles of Agile, leading to inefficiencies, low morale, and reduced value delivery.

Team Anti-Patterns

- **Hero Culture:** Relying on one or two individuals to consistently overwork and "save the day." This leads to burnout and creates a single point of failure.
- **Silos:** Team members operate in functional silos (e.g., "front-end," "back-end," "QA") instead of collaborating as a cross-functional unit. This creates bottlenecks and slows down work.
- **Scrum Master as a Scribe:** The Scrum Master acts as a team secretary, taking notes and managing the board, instead of facilitating, coaching, and removing impediments.

Process Anti-Patterns

- **Sprint Zero:** An initial Sprint dedicated solely to setup, architecture, and planning without delivering any user value. Agile advocates for doing this work incrementally.
- **Standups as a Status Report:** The Daily Standup becomes a status report to a manager instead of a planning session for the Developers to synchronize their work.
- **Weaponized Metrics:** Using metrics like velocity to pressure a team, compare teams against each other, or measure individual performance. Velocity is a forecasting tool for the team, not a productivity metric.
- **Stretching the Sprint:** Extending the Sprint deadline when the work isn't finished. This breaks the consistent rhythm of Scrum and hides underlying issues.

Product Anti-Patterns

- **Absent Product Owner:** The Product Owner is not available to the team to answer questions, leading to delays, guesswork, and building the wrong thing.
- **Feature Factory:** The team focuses on shipping a high volume of features ("output") without measuring whether those features deliver any real value to the user ("outcome").
- **Proxy Product Owner:** A middleman (like a project manager) is placed between the development team and the true Product Owner, creating a communication bottleneck.

Daily Standup Best Practices

1. Keep it Consistent
Consistency is key to making the Daily Standup a routine habit.

- **Same Time, Same Place:** Hold the meeting at the same time and place every day. This reduces complexity and helps build a rhythm.

2. Timebox it to 15 Minutes
The meeting should be brief and to the point. The 15-minute timebox keeps the discussion focused and prevents it from turning into a long problem-solving session.

- **Action**: Use a timer and be diligent about sticking to the time limit.

3. It's for the Developers
This meeting is a planning session for the Developers, by the Developers. It is not a status report for the Scrum Master, Product Owner, or any other managers.

- **Action:** The Developers should run the meeting themselves. The Scrum Master's role is to ensure the meeting happens and is effective, but not to lead it.

4. Focus on the Sprint Goal
The primary focus of the meeting should be on the team's collective progress toward the Sprint Goal, not just individual tasks.

- **Action:** Instead of just listing tasks, frame your updates in the context of the goal. For example, "Yesterday, I finished the payment validation, which moves us closer to our goal of a working checkout process."

5. Walk the Board Kanban
A highly effective practice is to structure the meeting around the work, not the people.

- **Action:** Instead of going person by person, move across your team's board from right to left (from the columns closest to "Done" to the ones on the left). This keeps the focus on finishing work, not just starting it.

6. Take Problem-Solving Offline
The Daily Standup is for identifying problems, not solving them.

- **Action:** If a detailed discussion is needed to solve a problem, schedule a separate, follow-up meeting with only the necessary people. This respects everyone's time and keeps the standup on track.

Kanban WIP Limits Playbook

A **Kanban WIP Limits Playbook** is a guide to implementing and using one of the most critical practices in Kanban: limiting your **Work in Progress (WIP)**. WIP limits are the engine of a Kanban system, creating a "pull" system that improves flow and exposes bottlenecks.

Step 1: Visualize Your Workflow
Before you can set limits, you must have a clear, visual representation of your team's workflow on a Kanban board.

- **Action**: Define the columns on your board to represent each distinct stage of your process, from when work is started to when it's completed (e.g., "To Do," "In Development," "Testing," "Done").

Step 2: Set Your Initial WIP Limits
Start simple. Don't try to set a limit for every single column at the beginning.

- **Action**: Set an overall WIP limit for all of your "in-progress" columns combined. A common starting point is **1.5 times the number of people on your team**.
- **Example**: For a team of 4 people, a good starting WIP limit would be 6 (4 x 1.5 = 6). This means there should never be more than 6 items being actively worked on at one time.
- **Visualize the Limit:** Place this number at the top of your "In Progress" section.

Step 3: The "Stop the Line" Play
This is the most important rule. When your WIP limit is reached, the team's priority shifts from starting new work to finishing existing work.

- **Action:** If the number of items in your "In Progress" columns equals the WIP limit, **the team is not allowed to pull any new work.**
- **The Play:** The team's entire focus should now be on unblocking the existing work and moving items to "Done." This "stop the line" approach is what creates a smooth, continuous flow.

Step 4: Identify Bottlenecks
WIP limits will quickly make your process bottlenecks visible.

- **Action:** Observe your Kanban board. The column where work consistently piles up is your bottleneck. For example, if your "Testing" column is always full, you have a bottleneck in your testing process.

Step 5: Adjust and Refine Your Limits
Your initial limits are a starting point. The goal is to continuously adjust them to improve your workflow.

- **Action:** Once you've identified a bottleneck, you can refine your limits. You might lower the overall WIP limit to reduce multitasking, or you could add a specific WIP limit to the bottleneck column to prevent it from getting overloaded.
- **Example:** For the team of 4, you might adjust from an overall limit of 6 to specific limits: "In Development" (WIP Limit: 4) and "Testing" (WIP Limit: 2).

Team Working Agreement Template

A **Team Working Agreement** is a living document created and agreed upon by the entire team. It outlines a set of norms and expectations for how the team will work together to achieve its goals. Its purpose is to foster a positive, productive, and psychologically safe environment.

Here is a template your team can use to create its own working agreement.

Team Working Agreement Template
Team Name: *[Insert Team Name]*
Date Created: *September 29, 2025*

1. Our Core Values
What principles are most important to us as a team?

- Example:
  - **Respect:** We listen to understand, not just to respond.
  - **Transparency:** We are open and honest in our communication.
  - **Collaboration:** We succeed or fail as a team.

2. Communication Norms
How, when, and where do we communicate?

- **Primary Communication Channel (e.g., Slack):** For quick questions and daily updates.
- **Formal Communication (e.g., Email):** For communication with external stakeholders.
- **Core Working Hours:** We will be available and responsive between *[e.g., 10 AM and 4 PM]*.
- **Response Times:** We will acknowledge non-urgent messages within *[e.g., 4 hours]*.

3. Meeting Guidelines
How do we ensure our meetings are effective?

- **Agendas:** All meetings must have a clear agenda sent out in advance.
- **Punctuality:** We start and end all meetings on time.
- **Participation:** We encourage everyone to contribute, and we will be mindful of not interrupting others.

4. Ways of Working
What are our agreements on how we get work done?

- **Definition of Done:** *[Link to or list your team's Definition of Done]*
- **Code Reviews:** All code requires at least one approval before merging. Feedback should be constructive and timely.
- **Focus Time:** We will respect "no meeting" blocks on calendars to allow for deep work.

5. Decision Making
How will we make decisions as a group?

- **Our Approach:** For most decisions, we will use *[e.g., consensus, majority vote, or consult with the Tech Lead]*.

6. Conflict Resolution
How will we handle disagreements?

- **Our Process:** We agree to address conflicts directly and respectfully with the person involved first. If a resolution can't be reached, we will involve the *[e.g., Scrum Master or Manager]* to facilitate.

Team Agreement
By signing below, we agree to uphold these principles and hold each other accountable.

- *[Team Member 1 Name]*
- *[Team Member 2 Name]*
- *[Team Member 3 Name]*

Service ROCKET

The Future of Atlassian Freelancing Starts Here

The hardest part of freelancing isn't the work. It's getting into the big projects. Compliance, contracts, client access. The barriers add up.

That's why we built **Crew Member by ServiceRocket**. It's more than a program, it's a community.

As a Crew Member, you'll get access to larger opportunities, support for your certifications, and a global network of Atlassian experts. For some, it even becomes the first step toward entrepreneurship.

More than a contract. It's your chance to grow, connect, and thrive.

Grow your freelance career.

Connect with global Atlassian opportunities.
Begin at crew.servicerocket.com

Stakeholder Communication Guide

Step 1: Identify and Analyze Your Stakeholders
The first step is to identify who your stakeholders are and then categorize them to determine how to engage with them. A common tool for this is the Power/Interest Grid.

- **Action:** List all your stakeholders and place them into one of the four quadrants below.

- **High Power / High Interest (Manage Closely):** These are key players. You need to fully engage with them and make the greatest efforts to satisfy them.
- **High Power / Low Interest (Keep Satisfied):** Put enough work in with these people to keep them satisfied, but not so much that they become bored with your message.
- **Low Power / High Interest (Keep Informed):** Adequately inform these people, and talk to them to ensure that no major issues are arising.
- **Low Power / Low Interest (Monitor):** Monitor these stakeholders, but do not bore them with excessive communication.

Step 2: Create a Communication Plan Matrix
Once you've categorized your stakeholders, create a simple plan that outlines how you will communicate with each group.

- Action: Fill out a matrix like the one below to create a clear and actionable plan.

Stakeholder Group	Communication Goal	Key Message	Channel	Frequency
Example: Project Sponsor	Gain approval, ensure confidence	Progress against goals, key risks	1:1 Meeting	Weekly
Example: Executive Leadership	Provide high-level status	KPI Dashboard, key milestones	Email Summary	Monthly
Example: End Users	Gather feedback, provide updates	New features, upcoming changes	Newsletter	Bi-weekly
Example: Marketing Team	Align on launch activities	Release timeline, feature benefits	Sync Meeting	Weekly

Step 3: Execute and Adapt
Your communication plan is a living document. The final step is to put it into action and be prepared to adapt.

- Be Consistent: Stick to the cadence you've defined. Consistent communication builds trust and predictability.
- Be Clear and Concise: Tailor your message to your audience. Executives need a high-level summary, while a development team may need technical details.
- Listen and Gather Feedback: Communication is a two-way street. Actively listen to your stakeholders' feedback and concerns.
- Adapt Your Plan: Periodically review your communication plan. If a certain channel isn't working or if a stakeholder's needs change, be ready to adjust your approach.

Risk Management in Agile

1. Continuous Risk Management Through Agile Events
Agile frameworks like Scrum have built-in mechanisms for managing risk in every Sprint.

- **Sprint Planning:** The team discusses risks related to the work they are committing to, such as technical uncertainty or dependencies.
- **Daily Scrum:** This is a daily risk identification meeting. When a team member raises an impediment, they are identifying a risk to the Sprint Goal.
- **Sprint Review:** By demonstrating a working increment of the product to stakeholders every Sprint, the team mitigates the single biggest risk: building the wrong thing.
- **Sprint Retrospective:** The team discusses and addresses process-related risks, such as communication issues or technical debt, to improve their workflow.

2. Using the Backlog as a Risk Mitigation Tool
The Product Backlog is a powerful tool for managing risk.

- **Prioritize Risky Work First:** The Product Owner can prioritize technically challenging or uncertain user stories to be worked on early. This "risk-first" approach ensures that the biggest unknowns are tackled when the team has the most time to deal with them.
- **Use Spikes:** When there is a significant technical or design uncertainty, the team can create a Spike. A Spike is a time-boxed research task that is added to the backlog. Its goal is to gain knowledge and reduce uncertainty, thereby mitigating the risk.

3. The ROAM Technique for Risk Management
A simple yet effective way to manage identified risks is the **ROAM** technique. When a risk is identified, the team collaboratively decides which category it falls into:

- **R - Resolved:** The risk is no longer a concern and has been addressed.
- **O - Owned:** Someone on the team has taken ownership of managing the risk.
- **A - Accepted:** The risk is understood, and the team has decided to accept it without taking any action at this time.
- **M - Mitigated:** The team creates a plan to reduce the probability or impact of the risk.

4. Transparency as the Best Defense
The most effective risk management strategy in Agile is a culture of transparency and psychological safety. When team members feel safe to raise concerns early, risks are identified when they are small and easy to manage, rather than after they have become major problems.

Agile Mindset & Culture Guide

The Four Pillars of an Agile Mindset

1. Customer Collaboration over Contract Negotiation
An Agile culture prioritizes building a partnership with the customer. The focus is on a continuous feedback loop to ensure the team is building the right thing.

- **Fixed Mindset:** "We delivered what the contract specified."
- **Agile Mindset:** "We worked with the customer to solve their problem."

2. Responding to Change over Following a Plan
Change is not seen as a disruption but as an opportunity to learn and deliver more value. Agile teams expect and welcome change, even late in the development cycle.

- **Fixed Mindset:** "We must stick to the original plan."
- **Agile Mindset:** "How can we adapt to this new information?"

3. Individuals and Interactions over Processes and Tools
While processes and tools are important, an Agile culture values the people doing the work and their ability to collaborate effectively. Empowered, self-managing teams are trusted to find the best way to get the work done.

- **Fixed Mindset:** "Did you follow the process?"
- **Agile Mindset:** "Did you collaborate with your team to solve the problem?"

4. Working Software over Comprehensive Documentation
The primary measure of progress is the delivery of a working, valuable product. While documentation has its place, it is secondary to delivering functionality that solves a user's problem.

- **Fixed Mindset:** "Is the documentation complete?"
- **Agile Mindset:** "Does the software work and deliver value?"

Key Cultural Traits

An Agile mindset fosters a specific set of cultural attributes:

- **Psychological Safety:** An environment where team members feel safe to experiment, ask questions, and fail without blame.
- **Continuous Improvement:** A relentless focus on getting better. The team regularly inspects its process and adapts through events like Sprint Retrospectives.
- **Focus on Value:** All work is prioritized based on the value it delivers to the customer, not on completing a pre-defined list of tasks.
- **Empowerment and Trust:** Leadership trusts the team to make decisions about how to do their work, fostering a sense of ownership and a

Forge Development

Setup Your Forge Developemnt Enviroment

Section 1: Prerequisites
Before we install the Forge CLI (Command Line Interface), you need a few things in place. Let's make sure your system is ready.

1. An Atlassian Account and Developer Site
You need an Atlassian account to log in and manage your apps. You also need a cloud development site (Jira or Confluence) where you can install and test your apps.

- **Action:** If you don't have one, sign up for a free Atlassian Cloud developer site. This will give you a real Jira and Confluence instance to work with. Visit: https://developer.atlassian.com/signup/

2. Node.js and npm
The Forge CLI is built on Node.js. It's distributed and installed via npm (Node Package Manager), which comes bundled with Node.js.

- **Action:** Download and install the latest LTS (Long-Term Support) version of Node.js from the official website: https://nodejs.org/
- **Verification:** Open your terminal or command prompt and run the following commands to ensure they are installed correctly:

  ```
  node -v
  npm -v
  ```

  You should see the version numbers printed for both.

3. Docker Desktop (optional)
Forge uses Docker containers to run your app in a secure, isolated environment that mirrors Atlassian's cloud infrastructure. The forge tunnel command, which is essential for local development, relies on Docker.

- **Action:** Download, install, and run Docker Desktop for your operating system (Mac or Windows). Visit: https://www.docker.com/products/docker-desktop/
- **Important:** Make sure Docker Desktop is running in the background before you start your Forge development session.

4. A Code Editor
While you can use any text editor, a modern code editor like Visual Studio Code will provide features like syntax highlighting, terminal integration, and extensions that make Forge development much easier.

- **Action:** Download VS Code from https://code.visualstudio.com/

Section 2: Installing and Configuring the Forge CLI
With the prerequisites out of the way, it's time to install the core tool: the Forge Command Line Interface (CLI).

Step 1: Install the Forge CLI
The CLI is an npm package. To install it globally on your system, open your terminal and run this command:

```
npm install -g @forge/cli
```

Note: The -g flag installs the package globally, making the forge command available anywhere in your terminal.

Step 2: Authenticate with Your Atlassian Account
Next, you need to connect the Forge CLI to your Atlassian account using an Atlassian API token.

1. Create the API Token:
- Navigate to https://id.atlassian.com/manage/api-tokens in your web browser.
- Click **Create API token**.
- Enter a memorable Label to describe your token, for example, forge-cli-dev-token.
- Click **Create**.
- Click **Copy to clipboard** and close the dialog. This is the only time you can view the token, so paste it somewhere safe temporarily.

2. Log in with the Forge CLI:
Now, go back to your terminal and run the login command:

```
forge login
```

- Enter the email address for your Atlassian account when prompted.
- When prompted for the API token, paste the token you just copied from the Atlassian site.

Section 3: Creating and Running Your First App
Let's build a simple "Hello World" app to confirm everything is working correctly.

Step 1: Create the App
The *forge create* command will generate a starter project from a template.
1. In your terminal, navigate to the directory where you want to store your projects.
2. Run the command:

```
forge create
```

3. You'll be prompted to:
    - Enter a name for your app: Let's call it hello-world.
    - Select a category: Choose UI Kit 2.
    - Select a template: Choose the jira-global-page template.

The CLI will create a new directory named hello-world with all the necessary files.

Step 2: Navigate into Your App Directory

```
cd hello-world
```

Step 3: Start the Development Tunnel (optional)
To see your app running and to enable hot-reloading for code changes, you need to use the forge tunnel command. Remember to have Docker running for this step.

```
forge tunnel
```

This command compiles your code and creates a secure "tunnel" from your local machine to your Atlassian cloud development environment. Any changes you make to your local files will be reflected in your Jira site almost instantly.

Step 4: Deploy and Install the App
Your app is running locally, but now you need to deploy its initial version and install it on your development site.

1. Deploy the app: This packages your code and sends it to the Forge platform.

```
forge deploy
```

2. Install the app: This registers the deployed app with your specific Jira or Confluence site.

```
forge install
```

3. You will be prompted to:
- Select a product: Choose jira.
- Enter your site URL: Enter the URL of your development site (e.g., your-site-name.atlassian.net).

Step 5: See Your App in Action!
Open your Jira development site. In the main navigation sidebar on the left, you should see a new item with the Atlassian logo labeled "hello-world". Click it, and you will see your first Forge app running live!

You have successfully set up your environment, created, deployed, and installed your first Atlassian Forge app. You are now ready to start building.

Forge Command Cheat Sheet

Daily Development Essentials

Command	Description
`forge create`	Create a new app from a template
`orge register`	Register an app you didn't create so you can run commands for it
`forge deploy`	Deploy your app to the cloud
`forge install`	Install your app on a site
`forge tunnel`	Start a tunnel to connect local code with the app running in the development environment
`forge logs`	View application logs
`forge lint`	Check the source file for common errors

Installation & Setup

Command	Description
`npm install -g @forge/cli`	Install the Forge CLI
`forge login`	Log in to Forge with your Atlassian account
`forge logout`	Log out of Forge
`forge whoami`	Show the currently logged-in user
`forge autocomplete install`	Configures autocomplete for the Forge CLI

Environments

Command	Description
`forge environments`	Manage app environments
`forge environments list`	View all environments for this app
`forge environments create <name>`	Create a new environment
`forge environments delete <name>`	Delete an existing development environment

Deployment & Installation

Description	Description
`forge deploy --environment <name>`	Deploy your app to the specify the environment (see your default environment by running forge settings list)
`forge install --environment <name>`	Install to a specific environment
`forge install --upgrade`	Upgrade an existing app installation
`forge uninstall`	Remove your app from a site

Development & Debugging

Command	Description
`forge lint --fix`	Attempt to automatically fix any issues encountered

Storage & Variables

Command	Description
`forge storage`	Manage app environment variables
`forge variables list`	List environment variables for your app
`forge variables set <key> <value>`	Set an environment variable
`forge variables unset <key>`	Remove an environment variable

Help

Command	Description
`forge help`	Show help for Forge CLI
`forge help <command>`	Show help for a specific command

Atlassian Forge YAML Study Guide

1. Purpose of the manifest.yml

- This is the blueprint of your Forge app.
- It tells Forge:
  - What features the app has (modules)
  - Where the code for those features lives (resources)
  - What permissions the app needs (permissions)
  - What environment it runs in (app/runtime)

2. Basic Structure

```
modules:
  # App features go here

resources:
  # Code file mappings go here

permissions:
  # Scopes, external fetch, and other permissions

app:
  # Runtime and metadata about the app
```

3. Modules

- Define what your app can do in Atlassian products.
- Common types:
  - jira:issuePanel – adds a panel to Jira work items view.
  - jira:customField – adds a custom field to Jira.
  - confluence:contentAction – adds an action button to Confluence pages.

Example:

```
modules:
  jira:issuePanel:
    - key: my-issue-panel
      function: panelFunction
      title: My Panel
      viewportSize: medium
```

4. Resources

- Connect module keys to your code files.
- Key = name in the YAML.
- Path = file in your project folder.

Example:

```
resources:
  - key: panelFunction
    path: src/panel.jsx
```

5. Permissions

- Scopes = what your app can do inside Atlassian.
- External fetch = allow calling APIs outside Atlassian.
- Must be exactly what you need — extra scopes can get your app rejected.

Example:

```
permissions:
  scopes:
    - read:jira-work
    - write:jira-work
  external:
    fetch:
      backend:
        - https://api.example.com
```

6. App Settings

- runtime – Node.js version, memory, architecture.
- id – Unique app identifier (generated when app is created).

Example:

```
app:
  runtime:
    name: nodejs22.x
    memoryMB: 256
    architecture: arm64
  id: ari:cloud:ecosystem::app/your-app-id
```

7. YAML Syntax Tips

- Use two spaces for indentation (never tabs).
- Lists use - followed by a space.
- Keys are case-sensitive.
- Strings with special characters (:, #, ?) should be quoted:

Understanding Forge Environments

1. Development Environment

- This is your default environment when you create and deploy your app.
- You work here by running commands like:

```
forge deploy
forge install
```

- This environment is private to you and lets you test new code quickly.

2. Creating a Staging Environment

- Use staging to share your app with testers or a small group before going live.
- To create a staging environment, run:

```
forge environment create staging
```

- You can then deploy your app to staging with:

```
forge deploy --environment staging
```

- And install the app in a test site (or Jira instance) for your team:

```
forge install --environment staging
```

3. Creating a Production Environment

- Production is the live version your users will see.
- Create it with:

```
forge environment create production
```

- Deploy your stable, tested app:

```
forge deploy --environment production
```

- Install it for your real users:

```
forge install --environment production
```

Switching Between Environments

- When running forge deploy, forge install, or other commands, use --environment <env> to specify the target.
- For example, to deploy to staging:

```
forge deploy --environment staging
```

Why Use Multiple Environments?

- Safety: Avoid breaking your live app with unfinished code.
- Testing: Try features in staging with real data and workflows.
- Rollbacks: If production has issues, you can switch back to an earlier environment or version.

UI Kit vs Custom UI Comparison

Quick Summary

- **Custom UI** = total control + complexity + isolated iframe environment.
- **UI Kit** = quick start + consistent Atlassian look + simpler integration.

	UI Kit	**Custom UI**
What it is	Use pre-built Atlassian UI components provided by Forge's UI Kit library	Build your own UI from scratch using standard web tech (React, HTML, CSS, JS) inside an iframe
Development	Faster to build; components are ready-made and styled consistently with Atlassian	More flexible; you control every detail of the interface, styles, and libraries
Performance	Lightweight; uses native rendering inside Jira/Confluence	Can be heavier due to iframe and full frontend bundle
Communication	UI and backend run closer together, direct API calls without iframe	Uses @forge/bridge for messaging between iframe UI and backend
Styling	Limited to UI Kit's theming and components	Full control; use any CSS or UI framework
Use cases	Standard forms, dialogs, lists, simple UI aligned with Atlassian design	Complex, unique UI needs; custom visualizations; third-party libraries
Security	Runs as part of Atlassian UI with Forge permissions	Runs sandboxed inside iframe
Setup complexity	Easier, no separate build or static file serving required	More setup (serving static files, building frontend)
Maintenance	Less frontend maintenance due to UI Kit updates managed by Atlassian	You maintain all frontend code and dependencies

UI Kit

What UI Kit is

- **Purpose:** UI Kit is Forge's built-in set of ready-made UI components that are simple, lightweight, and render natively in the host product (like Jira or Confluence).
- **Why it's special:** The UI isn't HTML/CSS — it's defined with Forge's @forge/ui components, and Forge handles rendering it in Atlassian's style.
- **Key benefits:**
  - **Fast to build** — you don't write CSS or HTML.
  - **Secure** — no custom JavaScript execution, so there's less risk.
  - **Native look & feel** — matches Jira/Confluence automatically.

How your manifest shows it's UI Kit

```
modules:
  jira:issuePanel:
    - key: ui-kit-demo-hello-world-issue-panel
      resource: main
      resolver:
        function: resolver
      render: native
      title: UI Kit Demo
      icon: https://developer.atlassian.com/images/icons/issue-panel-icon.svg
```

- render: native → tells Forge this module uses UI Kit.
- jira:issuePanel → means the UI will appear in Jira work items as a panel.
- title & icon → label and icon shown in the UI.

Marketplace App Submission Checklist

I. Foundational & Manifest (manifest.yml) Setup

- [ ] Create an Atlassian account and register as a Marketplace vendor.
- [ ] In manifest.yml: Define a globally unique app key and name.
- [ ] In manifest.yml: Add your vendor name and a link to your company homepage under the vendor key.
- [ ] In manifest.yml: Add a compelling description and a summary for the app listing.
- [ ] In manifest.yml: Define necessary scopes (permissions) following the principle of least privilege.
- [ ] In manifest.yml: Configure app icons with correct paths to your 16x16 and 128x128 PNG assets.
- [ ] For paid apps: Enable licensing by adding licensing: paid to your manifest.yml.
- [ ] Run forge settings set-environment production before the final deployment to ensure you are not on a development environment.
- [ ] Deploy the app to the production environment using forge deploy -e production.

II. Technical & Security Requirements (Forge Specific)

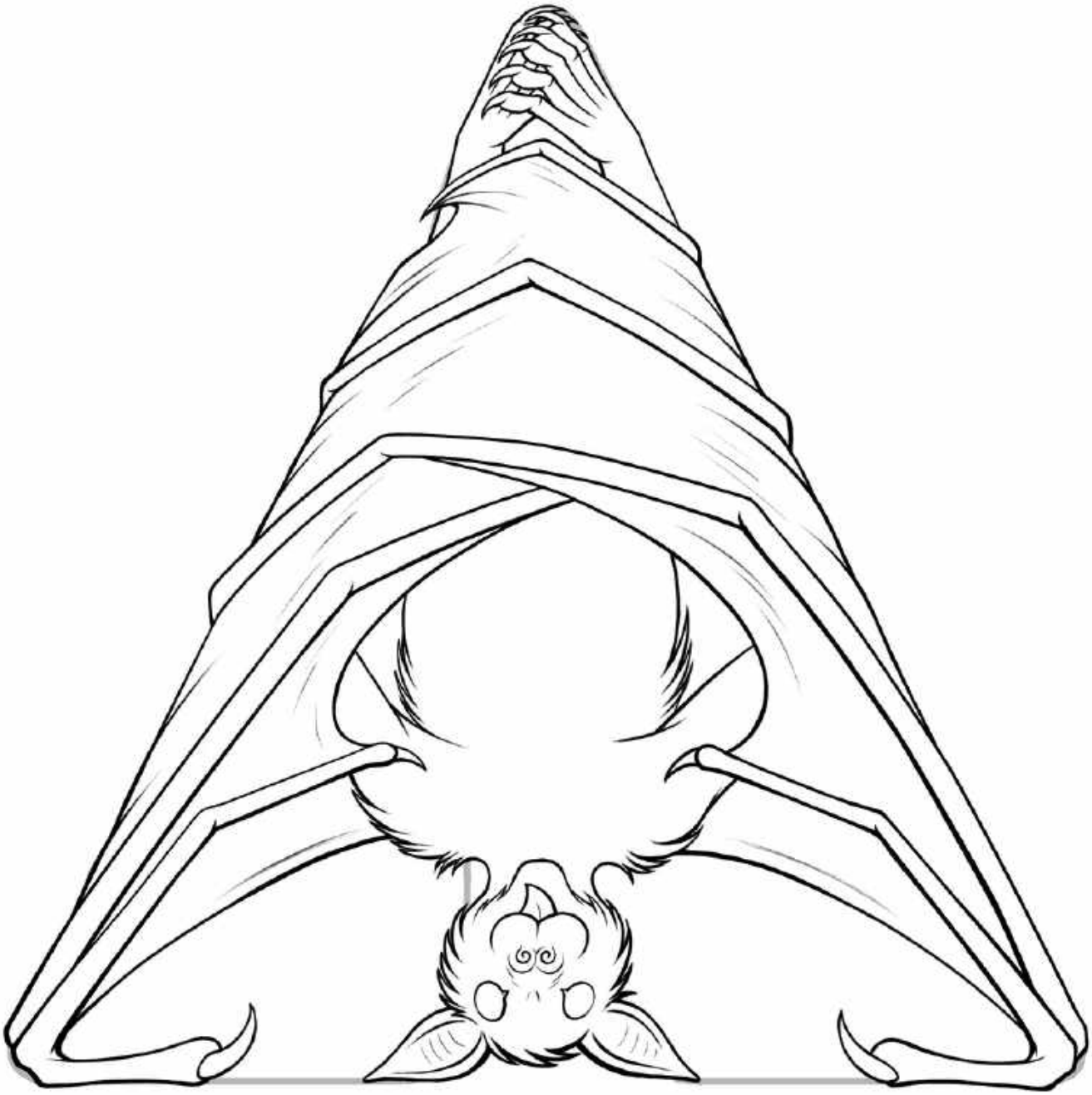
- [ ] Test all app functionality thoroughly in the production environment.
- [ ] Verify your app operates within Forge platform limits (e.g., invocation limits, storage quotas, egress permissions).
- [ ] If using external services, configure and test egressPermissions in the manifest to ensure connectivity.
- [ ] Securely manage secrets and API tokens using Forge's environment variables (forge variables). Do not hardcode secrets.
- [ ] Review and sanitize all user inputs to prevent injection attacks (relevant for both UI Kit and Custom UI).
- [ ] If using Custom UI, ensure all third-party libraries are up-to-date and free of known vulnerabilities.

III. Listing, Documentation & Support

- [ ] Create high-quality screenshots and/or a short video demonstrating your app's functionality.
- [ ] Prepare a comprehensive user guide explaining how to use your app's features.
- [ ] Provide a clear and working link to your support channel (e.g., support portal, email).
- [ ] Write and host a public Privacy Policy and link to it in your Marketplace listing.
- [ ] Establish a process to handle data subject requests (e.g., for data access or deletion) as outlined in your Privacy Policy.

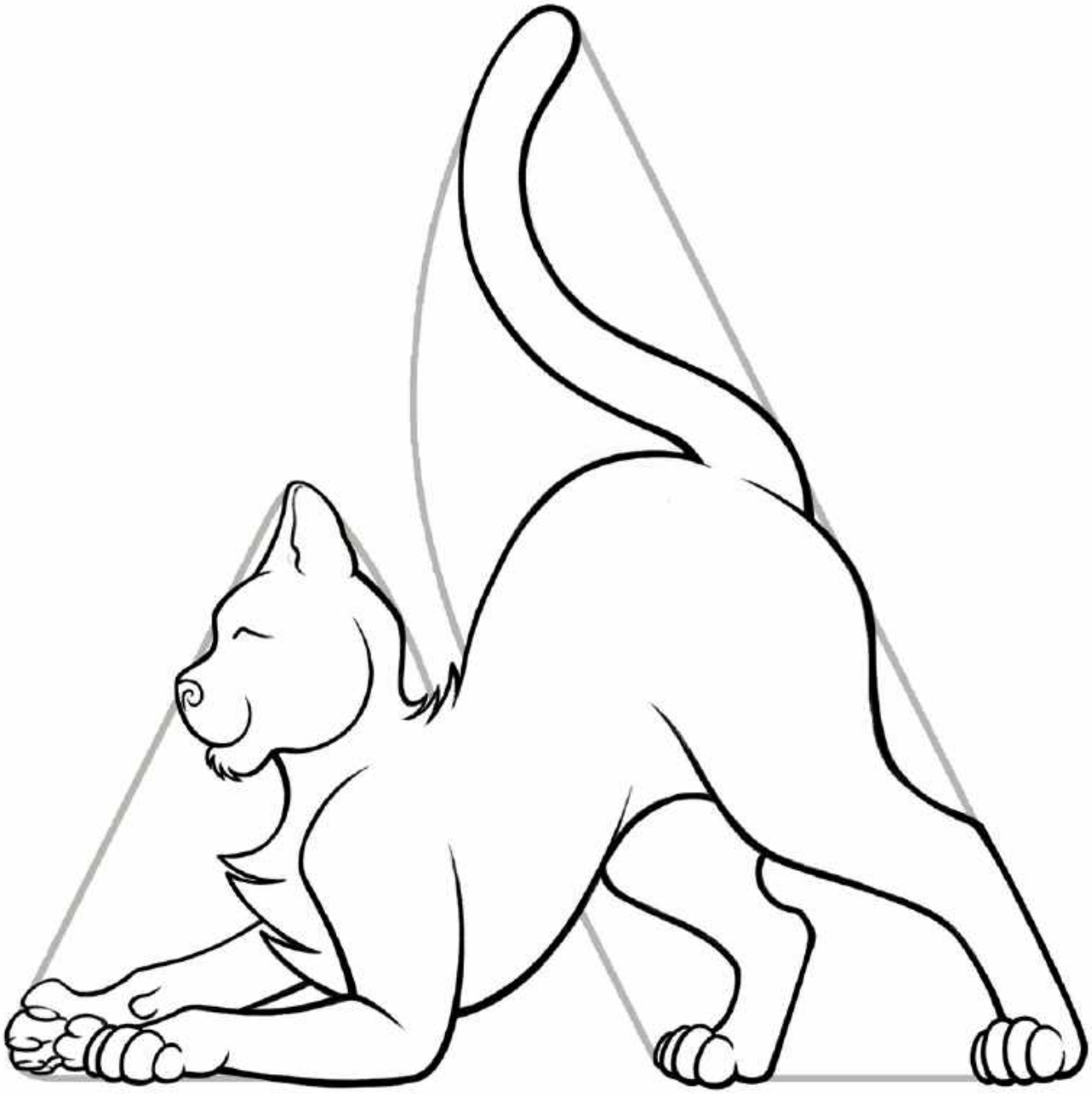
IV. Legal & Pre-Submission Final Review

- [ ] Provide an End-User License Agreement (EULA) or opt to use the standard Atlassian EULA.
- [ ] Review and accept the Atlassian Marketplace Partner Agreement.
- [ ] Ensure your app name, logo, and marketing materials comply with Atlassian's brand guidelines.
- [ ] Double-check that all support, documentation, and privacy policy links are public and working correctly.
- [ ] Confirm your app provides clear value and functionality as described in the listing.



BATLASSIAN

by Kythera Contreras



CATLASSIAN

by Kythera Contreras

by Kythera Contreras

Notes

Notes

Notes

Notes

Notes

CollabSoft VENTURES

Funding the next generation of Atlassian Marketplace developers

- Fast loans of $10K—$100K
- Founder-friendly
- Easier than traditional funding
- Full autonomy
- No oversight

Got a great app idea?
We'll back you, not own you.

Find out more at:

collabsoft.com/collabsoft-ventures

gryd.io

Helping Your Team Thrive With Atlassian

Get one actionable Jira tip.
Delivered to your inbox.
Every week.

www.gryd.io