exalate

Security and Architecture Whitepaper



ISO 27001:2022
Certified by
Brand Compliance
MGMT. SYS.
RvA C 548

Table of Contents

03 Introduction
- 04 ISO 27001 Compliance

05 Behind the Scenes of Exalate
- 05 The Synchronization Process
- 11 Deployment Models
- 21 Connection Types
- 22 Security Considerations
- 25 Authentication and Authorization Techniques

26 Exalate Console
- 26 Exalate Classic Console and New Exalate Console
- 31 Exalate Classic Console
- 37 The New Exalate Console

46 Exalate AI
- 46 Evolution of Exalate AI
- 48 How are the AI Services Being Accessed

51 Exalate Cloud
- 51 Security Components
- 52 Access to The Environment
- 53 Where Is Your Data Stored?

54 Application Security
- 54 Security By Design
- 54 Atlassian Cloud Fortified
- 55 Penetration Testing
- 55 Other Application Security Measures
- 56 Security Best Practices: AI Assist

57 Security FAQs

58 Glossary

1. Introduction

Companies use a variety of applications like work (task) management systems, ITSM tools, CRMs, project management tools, etc. This whitepaper provides an overview of how Exalate handles information exchange between these multiple platforms.

In any type of information exchange, it is best to ensure it remains secure so that:

1. Information is exchanged between all known parties.
   a. The sender wants to be sure that the information is sent to the right destination.
   b. The recipient wants to be sure that the information is received from the right source.
2. The exchanged information has not been tampered with by third parties.
3. The information is restricted to what the sender wants to send, and what the receiver wants to receive.
4. Every data exchange is logged for audit purposes.
5. Any security breach is escalated to all stakeholders.

Exalate is **ISO 27001:2022 certified**. It has an advanced Groovy-based synchronization engine and various deployment models and information exchange paths that allow users to synchronize information between multiple platforms in a secure and controllable way.

To understand the security aspects of the solution, it is important to grasp how information is processed by what components, and how information flows through these components.

Note:

Learn more about How Exalate works and all the supported integrations here.

1.2 ISO 27001:2022 Compliance

Exalate's commitment to ISO 27001:2022 certification demonstrates adherence to an internationally recognized standard for information security management.

By implementing ISO 27001 controls, we ensure:

- **Access Control:** Restricting access to sensitive information, ensuring only authorized users interact with Exalate.
- **Risk Management:** Ongoing risk assessments to identify potential vulnerabilities and mitigate risks.
- **Auditing and Monitoring:** Regular audits are conducted to ensure compliance with ISO 27001.
- **Incident Response:** Procedures are in place to quickly respond to security incidents, minimizing the impact on data security and system integrity.

SOME USEFUL RESOURCES:

Exalate Information Security Policy

Statement of Applicability

2. Behind the Scenes of Exalate

How Exalate works and what components are at play?

2.1 The Synchronization Process

2.1.1 The Exalate Node

The Exalate Node is the application that acts as a gateway between an instance and the network connecting different environments directly. It has the following functions:

- Detecting that the data “under sync” has been changed or new data needs to be brought “under sync”
- Extracting the relevant data from the underlying system and formatting it as a structured message
- Exchanging the data with other nodes connected to the network
- Applying incoming messages locally

In the synchronization process, having autonomous control over integration involves four main components: two platforms that host the issues (or entities), and two Exalate Nodes.

BOOK A DEMO

2.1.2 Decentralized Integration

The most important feature of Exalate is “Decentralized integration” i.e. having independent control over the integration.

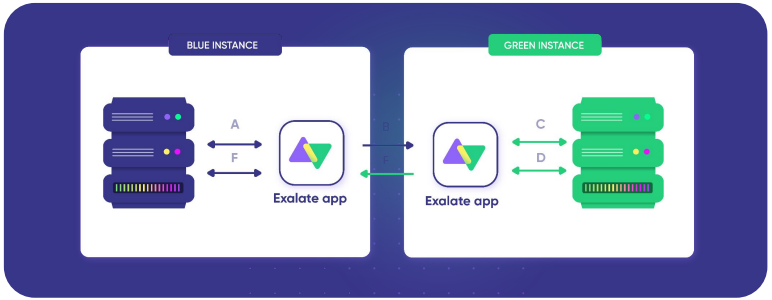
Whenever two teams decide to integrate their systems to exchange information, it must be done in such a way that neither of the parties lose control of what information is sent out and how incoming information is processed. This keeps the systems loosely coupled, thus ensuring scalability and maintainability.

This requirement is met because every side has its own Exalate Node which will translate any information between the local context and the Exalate network.

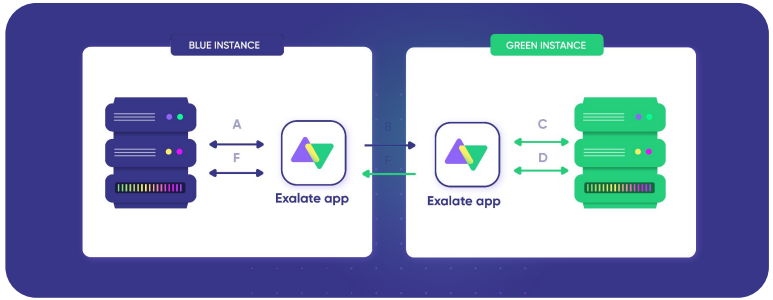
In the synchronization process, having autonomous control over integration involves four main components: two platforms that host the issues (or entities), and two Exalate Nodes.

The platforms can be any of these: Jira, Salesforce, ServiceNow, Freshdesk, Freshservice, GitHub, Zendesk, Azure DevOps (Cloud and Server), Asana, etc.

Find the complete list of supported integrations here.



The information flows as follows:



A. Exalate (Blue) interfaces with instance (Blue) for retrieving the entity (e.g. incident) information.

- It either polls the instance (Blue) for any changes since the last full synchronization or gets triggered by the instance (Blue) stating that a change has occurred (through a 'webhook').
- Then it collects all the relevant entity information and composes a message.

B. Exalate (Blue) sends the message to Exalate (Green) for further processing.

C. Exalate (Green) decomposes the message and applies the message to the instance (Green), such as updating an issue summary.

D. Exalate (Green) interfaces with instance (Green) to retrieve the changes that have been triggered by the synchronization and compose an acknowledgment message.

E. Exalate (Green) sends the acknowledgment message back to Exalate (Blue).

F. Exalate (Blue) processes the acknowledgment message and applies it to the instance (Blue).

2.1.3 The Synchronization Process Uses Queues which are Persisted in a Relational Database

There are many potential reasons why a synchronization might fail and therefore the whole process is asynchronous, using queues to go through every step of a synchronization transaction. Whenever a change happens to a set of information that requires synchronization, Exalate will pick up this change immediately and start to synchronize it.

It tracks the synchronization events, ensuring that every event processes in the same order as the original event. This means that all changes/updates will be queued and applied in the same order as it was made in the original entity. In case of an error, the transaction for that specific twin will be paused until the error is resolved. Exalate ensures that all changes are applied as soon as this error is fixed.

NOTE

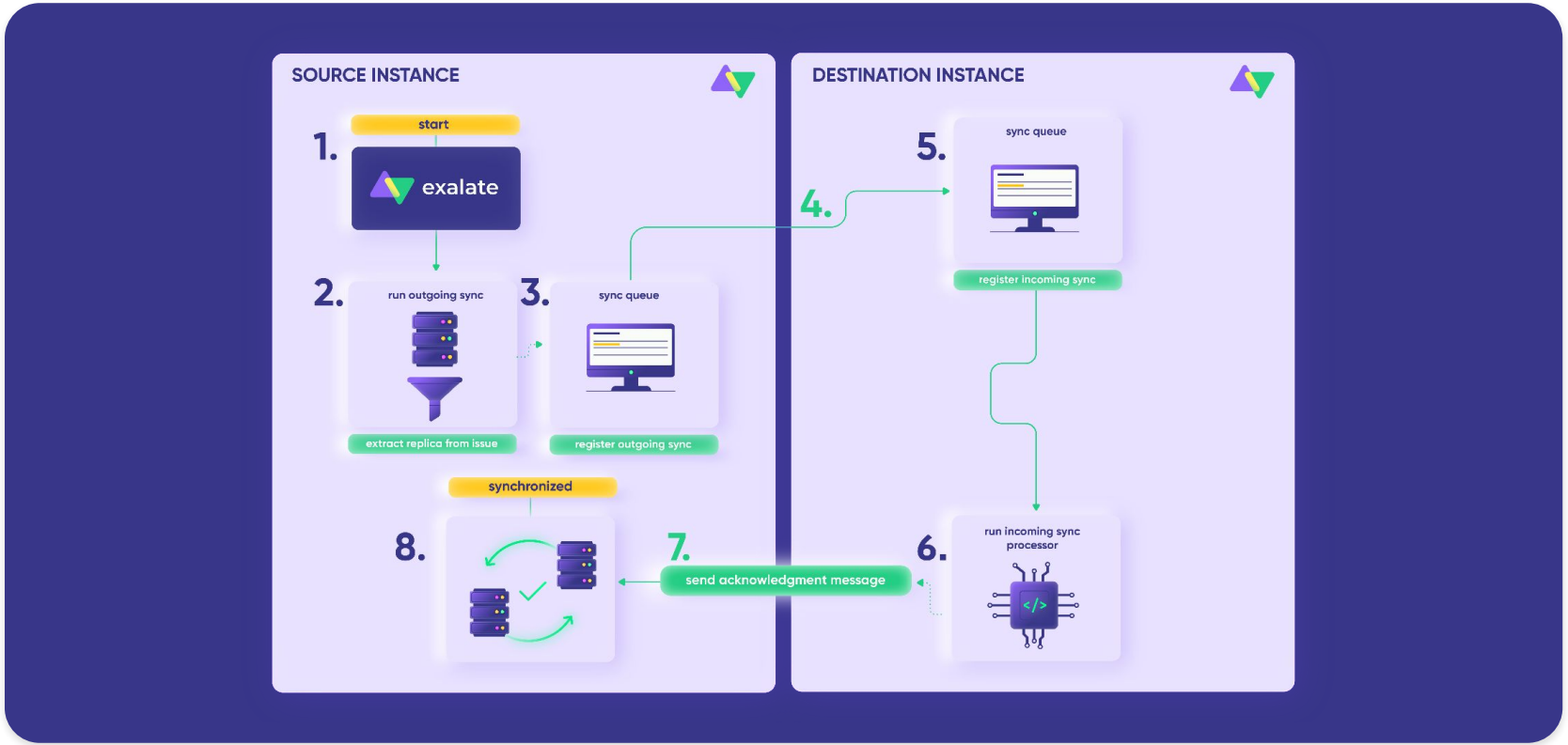
A twin is a pair of entities that are under sync. For example if you sync ISSUE-123 with incident 12345678, then the ISSUE-123 is the twin of incident 12345678.

While dealing with synchronization events, Exalate always only stores the most recent metadata to understand the relation between the synced entities.

Additionally, Exalate uses an integrated retry mechanism for resolving specific errors. It helps recover any failure and resume synchronization from the point of interruption.

NOTE

Immediately in the case of a push model where the instance is pushing the changes to the Exalate node using webhooks or similar type of technology. In case of polling, changes will be detected depending on the polling frequency, which is 5 secs by default if it is enabled.


SOURCE INSTANCE
DESTINATION INSTANCE
1.
start
exalate
2.
run outgoing sync
extract replica from issue
3.
sync queue
register outgoing sync
4.
5.
sync queue
register incoming sync
6.
run incoming sync processor
7.
send acknowledgment message
8.
synchronized

The numbers in the list below correspond to the numbers in the diagram:

1. The synchronization process starts with one of the available events: operation, trigger, API, post function, or through polling for change.

2. Exalate runs the outgoing sync processor and extracts a replica from the entity. With the outgoing sync processor, you can specify what information can be sent to the destination instance.

3. The outgoing sync event is registered in the Sync Queue.

4. The outgoing sync event is then sent to the destination instance.

5. The incoming sync request is registered in the Sync queue on the destination side.

6. Exalate runs the incoming sync processor with the received message and applies the change to the local entity.

7. Exalate runs the destination instance outgoing sync processor to create an acknowledgment message and sends this message to the source instance.

8. The act of synchronization is completed and all sync operation data is cleaned.

2.2 Deployment Models

Exalate's design goal is to provide a common information integration platform, independent of the particularities of the underlying system.

This section of the document outlines all the different deployment models supported by Exalate.

But before moving there, we should introduce the concept of "Single Tenancy".

2.2.1 Single Tenancy

Most cloud deployments are built using a multi-tenant architecture.

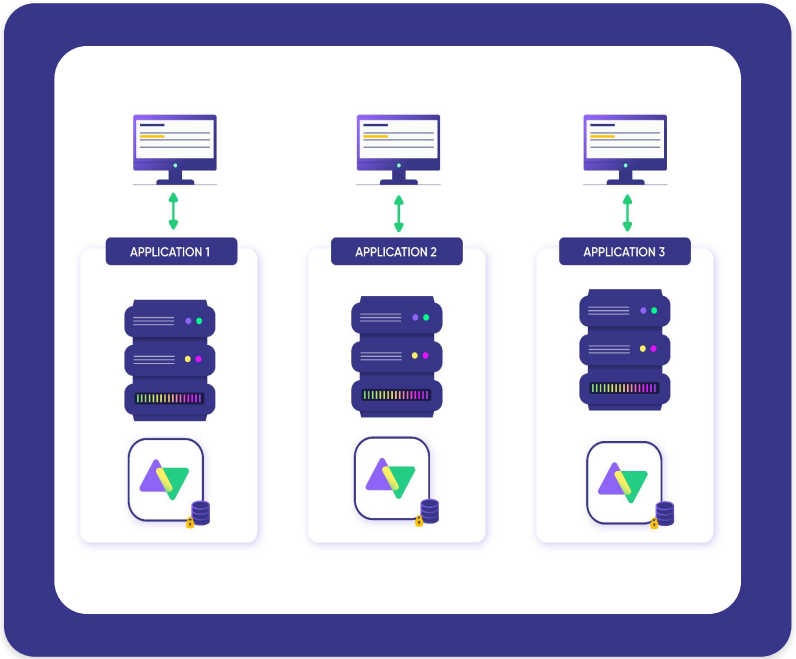
A multi-tenant architecture will use shared infrastructure to implement the required functionality. This means that the data of your competition is intermingled with your data at some level (database, file system, process, ...).



Single tenancy is the core feature that ensures data security.

Knowing that bugs can occur in systems, a multi-tenant architecture carries the risk that unwanted parties get access to your data.

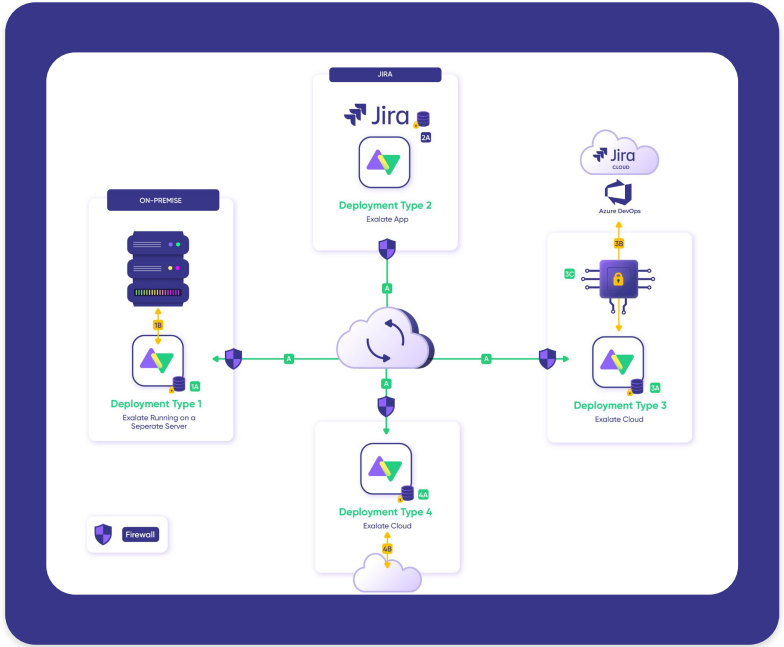
The Exalate engineers therefore chose to implement a single-tenant architecture. Due to this, a single instance of Exalate application and the corresponding environment support only one system. Every instance has its own process space, file system, network, and database completely separated from all other instances.



The model is available for all deployments of Exalate like Jira, Github, Freshdesk, Freshservice, Zendesk, Salesforce, ServiceNow, Asana, Azure DevOps Cloud and Server ...

2.2.2 Deployment Model Types

The diagram provides a full overview of how Exalate integrates with each of the different underlying platforms. Every deployment model has different information exchange paths.



2.2.3 Deployment Type 1

Exalate As a Separate Application On-Premise

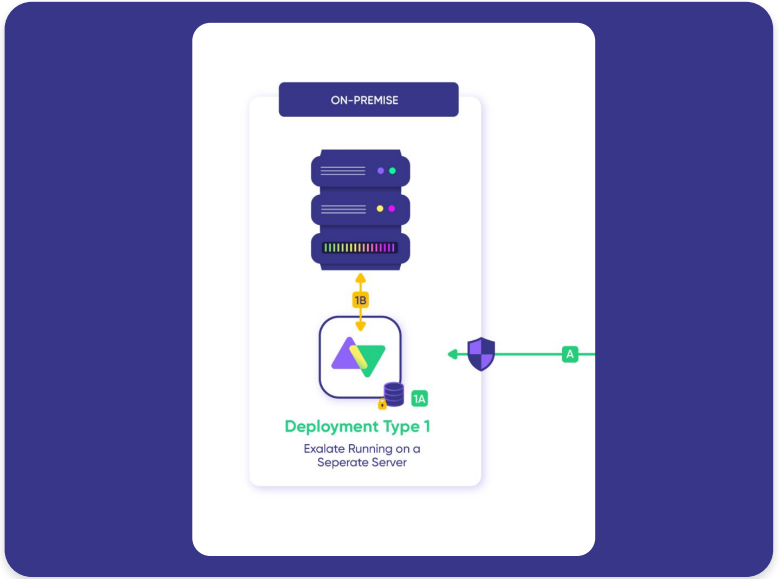
The ‘on-premise server’ deployment is used whenever you want to keep the integration processing within your security parameter. A separate (virtual) application server hosts the Exalate application, the database, and the file storage. The application acts as the intermediate gateway between the instance and the Exalate network

2.2.3.1 Connections

The connections are:

- Between Exalate and the instance on HTTP/S or custom ports as required by the platform, (Label 1B in Graph)
- Between Exalate and the other Exalate nodes (normally HTTP/S), (Label A in Graph)
- Between Exalate and the database (Postgres on port 5432), (Label 1A in Graph)

The server exposes port 9002 by default, which is used to access the Exalate console and the Exalate transport layer. These settings can be adapted and more information is available on our documentation site.



2.2.4 Deployment Type 2

Exalate As an App on Jira Data Center

The embedded application approach is used for Jira Data Center. Atlassian Jira provides an extension SDK, allowing you to run additional functionality in the same process space as the application itself. This allows for unrestricted access to any information maintained by Jira.



2.2.4.1 Connections

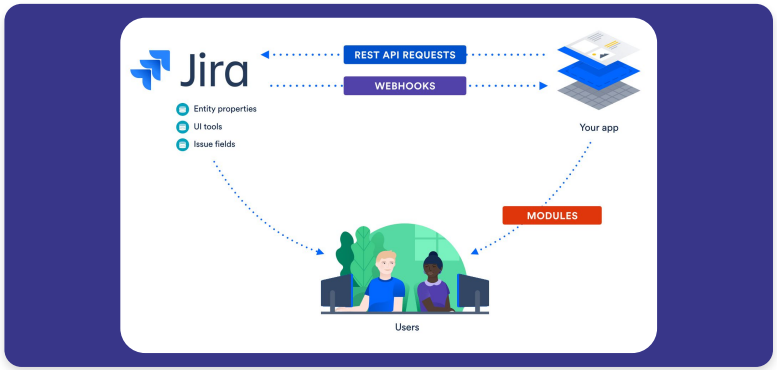
- Access to Exalate functionality, such as transport and console, is controlled by the Exalate REST API (Label A in the graph). The outgoing connections are between the Exalate app and the other Exalate nodes (normally HTTP/S).
- Access to the database is controlled by the 'Active Objects' database layer provided by the Jira environment (Label 2A).

2.2.5 Deployment Type 3

Exalate As an Atlassian Connect App

This deployment model is exclusively used for Jira Cloud (Exalate Classic), which requires Atlassian Connect apps.

These apps (add-ons) are web applications running on separate hosts. They are integrated with Atlassian Jira Cloud using various techniques such as REST API requests, webhooks, iframes, etc. The Exalate nodes are deployed in a single-tenant setup on the Exalate cloud.



The information available for the app is restricted by what the Jira Cloud API allows to access.

More information on the security measures between Jira Cloud and the app can be found here.

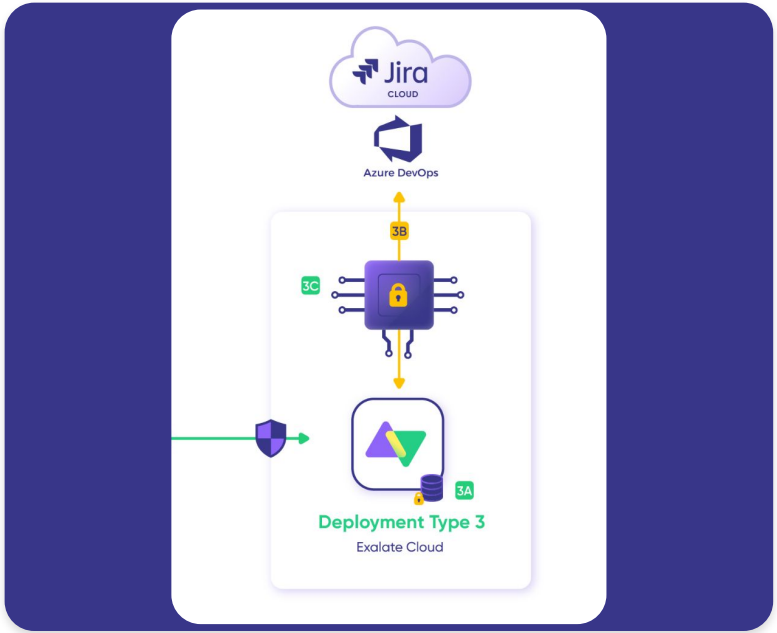
2.2.5.1 Single-Tenant Versus Multi-Tenant Setup - a Docker Primer

Atlassian Cloud apps are, in most cases, multi-tenant. This means that a single Atlassian Connect app will support multiple subscribing Atlassian applications. This is not an option for Exalate, as it provides a scripting interface that can execute database queries and REST calls.

It is essential that the environment provided for one customer **is completely separated** from the environment of another customer, as explained in section 2.2.1. The following components are essential to implement this single-tenant architecture:

- The docker-based node application: For each Jira Cloud instance, a separate environment is brought up, using advanced docker-related technologies. The process space, file system, and database are completely contained in a separate 'node-environment'. Exalate Cloud hosts multiple node environments.
- The mapper (Label 3C)
  Since Atlassian Cloud is interfacing with the application using a static URL (connect.exalate.net), there is a need to redirect these interactions to the correct node.

The mapper is a reverse proxy that will redirect calls from the Jira cloud to the right node application using the ID provided by Atlassian.



2.2.5.2 Connections of the Node Application

- Incoming connections Atlassian cloud connections redirected from the mapper (includes access to the console) (Label 3B) Exalate node connections for the synchronization message exchange (Label A)
- Outgoing connections Exalate node connections for the synchronization message exchange (Label A) Database access (Label 3A)

All nodes are fronted with a reverse proxy for SSL termination.

Connection to the database is on a docker-specific local network.

2.2.5.3 Mapper Connections

The mapper is a reverse proxy that accepts connections for

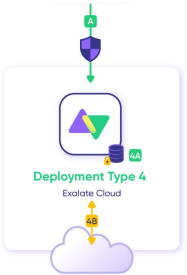
- Infrastructure management
- Atlassian cloud to Exalate app communication

2.2.6 Deployment Type 4

Exalate As a Separate Application in the Cloud

Exalate supports a number of cloud services, available for the supported platforms such as Jira, GitHub, Zendesk, Azure DevOps, Freshdesk, Freshservice, ServiceNow, Salesforce, Asetc. The main difference between this deployment model and the deployment model type 3 is that any connection with the tracker is direct, without a mapper in between.

The application itself is a docker environment operating in the same way as Jira Cloud nodes, where process space, file system, and database are completely separate from other environments.



2.2.6.1 Connections to the Instance (4A)

Push

Exalate will register with the instance to receive webhooks whenever the platform provides this capability - the instance will open a connection to the Exalate node to post the hook.

Poll

The poll model is used, whenever there is no push model possible. Through requests into the instance, Exalate identifies changes occurring on issues, incidents, tickets, or other entities - in this case, all connections are built from Exalate to the instance.

ALL TRAFFIC IS ENCRYPTED - this is enforced by the platforms.

NOTE

Backups are encrypted with Fernet symmetric encryption

2.2.6.2 Other Connections

For interactions with the instance, the Exalate node will communicate as follows:

- Incoming connections for console and synchronization traffic (Label A)
- Outgoing connections for synchronization traffic (Label A)
- Outgoing connections to the database server (Label 4B)

2.3 Connection Types

There are currently three types of connections you can set up for Exalate. This section outlines how they work. A connection type determines how connections between Exalate Nodes are built.

2.3.1 Public to Public

In a public-to-public connection type, TCP/IP connections are initiated from both sides.

It assumes there is an unobstructed network path between the two nodes. Synchronization events are transmitted to the other side from the moment these events occur.

2.3.2 Private to Public

This connection type is used whenever there is a need to connect a public instance, such as Jira Cloud/ Zendesk, with an internal server. A firewall obstructs any connections from the public/ Internet to the private server. In this case, all connections are built from the private server to the public server, collecting all changes that need to be applied locally.

To allowlist Exalate Cloud, use the IP addresses published at egress.exalate.cloud and ingress.exalate.cloud, as described in Egress and Ingress solution.

NOTE

It is also possible to deploy Exalate nodes on-premise and connect these without the need to provide access to the internet.



2.3.3 Local

Local connections are used to sync between local project configurations, such as between a Jira Service Desk (or Service Management) project and one or more Jira Software projects.

2.4 Security Considerations

Exalate is a middleware that transports data between two systems. This section focuses on how this data is protected by looking at confidentiality, integrity, and availability - also known as the CIA triad.

In section 2.1.3, we explained how data flows through the system. The data security aspects are relevant in the following steps:

Step 2 When the data is retrieved from the instance

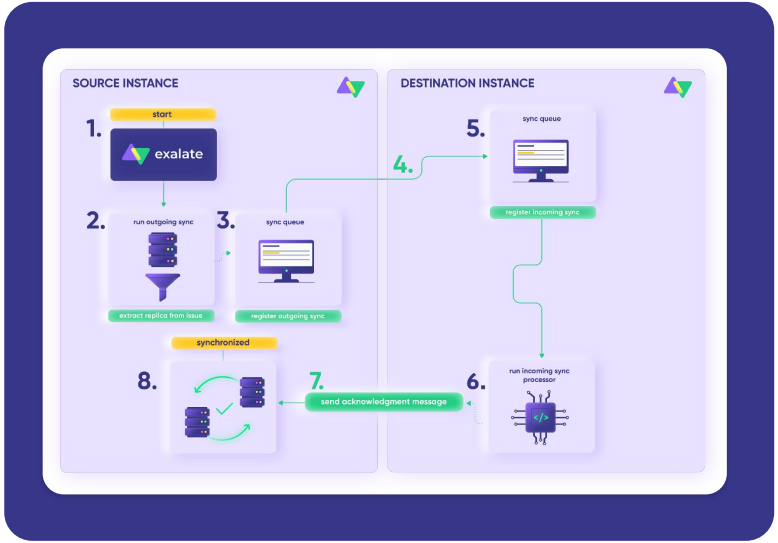
Step 3 When the data is stored as a sync event in the database

Step 4 a - When the data is sent to the other side
b - Where it gets queued into the sync requests queue

Step 5 When the data is processed and applied to the local entities

NOTE

Step 1, 6, and 7 are not mentioned on purpose.



The risks of a breach exist as follows:

Step 2 - Data retrieval

Risk:

- Man in the middle attack when the data is retrieved
- Data is retrieved from the wrong instance

Mitigation:

- For Deployment Types 1 and 3
  Used encrypted (HTTPS) traffic between the Exalate Node and the instance with properly defined certificates[1]
- For Deployment Model 3
  The communication between the instance and node is protected by an Atlassian-specific JWT mechanism - Click here for more information.

[1] Note that client-side certificates are not supported. As this is currently not supported by different trackers.

Step 3 - Enqueue on the sync event queue

Risk:

- Data is stored in the database and file system - this can be read when accessing the host.

Mitigation:

- Protect access to the hosts and the databases
- Encrypt the payload in the database

Step 4a - When the node sends the data to the other side

Risk:

- Man in the middle attack
- Information is sent to the wrong Exalate node
- Information is received from the wrong Exalate Node
- Configuration change in the database

Mitigation

- As explained in section 2.5.1, a JWT token mechanism is used to sign every message between the nodes such that the client is recognized as a trusted source.
- The destination risk is mitigated by ensuring that the connection is built using the invitation code..

Step 4b - Where it gets queued into the sync requests queue

Risk: See step 3

Mitigation: See step 3

Step 5 - When the data is processed and applied to the local issues

Risk: See step 2

Mitigation: See step 2

2.5 Authentication and Authorization Techniques

2.5.1 Source and Destination Authentication

One of the biggest risks in the exchange of information between a source Exalate Node and a destination Exalate Node is that the source sends data to the wrong destination, or the data is received from the wrong source. To mitigate this risk, a JWT-based approach has been implemented. JWT ensures that the client making the request is a trusted source.

For JWT to work, the client sending the HTTP(s) request needs to have a secret that the server also has. This acts as a private key to sign tokens with. Every request from the client must pass the token somewhere in the request - Exalate passes it in the "X-Exalate-JWT" header - so that the server can find the secret and validate the signature. If the signature proves that the client formed it using the same secret, the client has proven its identity.

This secret is exchanged at the moment that the connection is established and is contained in the invitation code.

2.5.2 Authorizations

Exalate requires multiple authorization sets for different purposes. To fully grasp these authorization sets it is important to take into account the relevant context and deployment type

2.5.3 Exalate Proxy User

The 'Exalate Proxy User' refers to the authentication context which is used to search, read and write the entities "under sync". As Exalate always uses the provided APIs of the underlying platform, the actual permissions have to be configured using the relevant permission approach provided by that platform. So the platform admin can define the required permissions the proxy user has.

For instance, if Exalate needs to add comments on a Jira Cloud, the permission 'Add comment' must be enabled. As long as the proxy user has the required permissions to create or update the entities "under sync", synchronization runs smoothly.

Going into the details of the permissions set is outside the scope of this whitepaper - more information is available on the documentation site.

3. Exalate Console

Unified visibility and governed control for complex integrations.

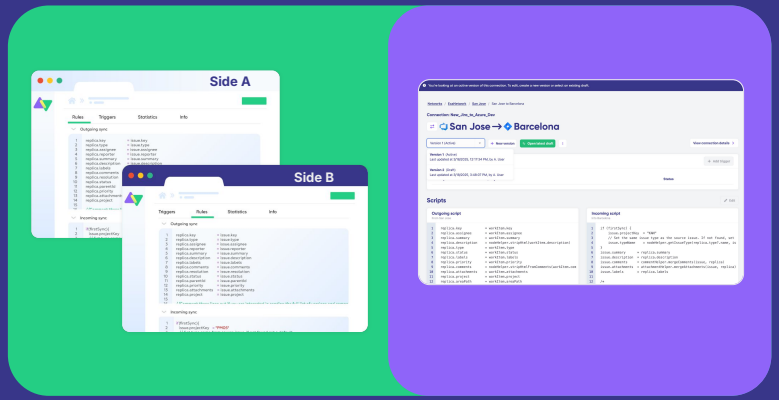
Integrations are no longer simple point-to-point connections. They have turned into large, interconnected environments spanning multiple tools, teams, and companies. As these environments scale, understanding what every integration does and changing it safely becomes as critical as the integration logic itself.

To address this reality, Exalate Classic evolved into the New Exalate experience. This evolution was necessary to align with modern integration requirements while enhancing operational capabilities that teams need at scale, all while preserving the autonomous control and security architecture that organizations depend on.

This ensures that as integration complexity grows, security and governance capabilities scale proportionally, maintaining the precision and independent decision-making required for cross-team collaboration at enterprise scale.

Exalate Classic Console

New Exalate Console



BOOK A DEMO

3.1 Exalate Classic And New Exalate: Differences and Similarities

Exalate Classic Console (Node-Level Administration)

Each Classic Exalate Node includes an embedded administration console accessible directly from the host platform (Jira, ServiceNow, Salesforce, etc.). Administrators configure connections, scripts, and triggers locally on each node.

Characteristics:

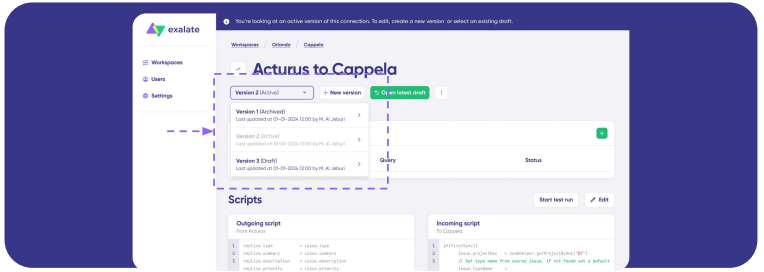
- Administration occurs per-node, requiring access to each platform instance
- Configuration changes applied locally; no cross-node visibility
- Suitable for simple peer-to-peer integrations with limited nodes
- Full scripting capability with local script management

New Exalate Console

The new Exalate console (exalate.app) provides unified oversight of all connected nodes from a single interface. Administrators manage connections, monitor network health, and configure synchronization rules across multiple nodes without accessing each platform individually.

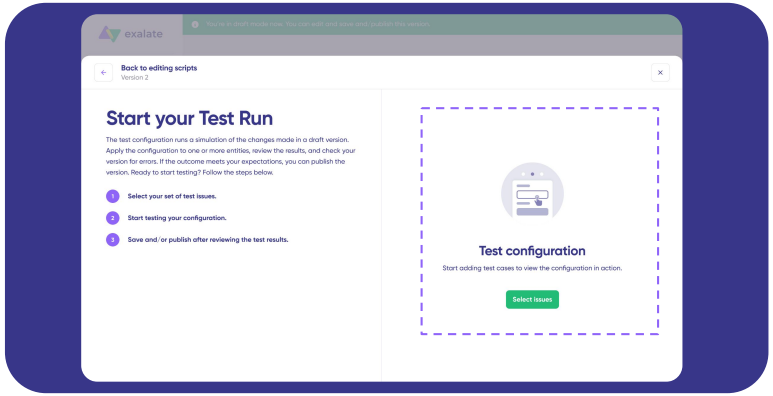
Characteristics:

- Single console for all nodes and connections across the integration network
- Visual integration topology displaying node (or connections) relationships
- Unified script versioning and Roll back
    - Traceable change history
    - Ability to revert mistakes
    - Safer configuration management



Script Versioning

- Test Run capabilities
  - Safely test your sync before deployment.
  - Reduce the risk of errors affecting live data.



Test Run

- Import existing connections from Exalate Classic Console for unified management
- Network-wide monitoring and troubleshooting

Exalate Classic and New Exalate: Use Cases

Scenario	Recommended
Single integration between two parties	**New Exalate Console**
Multiple nodes requiring unified oversight	**New Exalate Console**
Cross-company integrations with independent administration	**Exalate Classic Console (per party)**
Enterprise networks with centralized IT governance	**New Exalate Console**

NOTE

Both consoles remain available until all the use cases can be implemented using New Exalate.

3.1.1 User Directory and Console Authentication - a Comparison Between Exalate Classic and New Exalate

Exalate Classic: Tracker-Delegated Authentication

Exalate Classic maintains no internal user directory. Authentication is delegated entirely to the underlying task management system (Jira, ServiceNow, Azure DevOps, etc.). When administrators access the Exalate Classic Console, the host platform's native identity system validates credentials and authorization levels.

Characteristics:

- Authentication security inherits the host platform's identity management
- No cross-platform identity correlation: separate credentials per instance
- Console access tied to platform-specific administrator privileges
- MFA, password policies, and session controls depend on each instances' configuration

New Exalate: Centralized Identity via Keycloak

New Exalate uses Keycloak, an open-source identity and access management solution deployed within Exalate's infrastructure, as the centralized authentication point for all console access.

Architecture:

- Standards-based protocols: OpenID Connect (OIDC) and OAuth 2.0
- Google Social Login for authentication (Keycloak acts as OIDC relying party)
- Credential isolation: authentication occurs between user browser and Keycloak only
- No Google credentials stored; only identity tokens (email, name, profile identifier)

Session Security:

- Short-lived access tokens with refresh token rotation
- Token validation on each API request
- Sessions terminate on inactivity or explicit logout

3.2 Exalate Classic Console

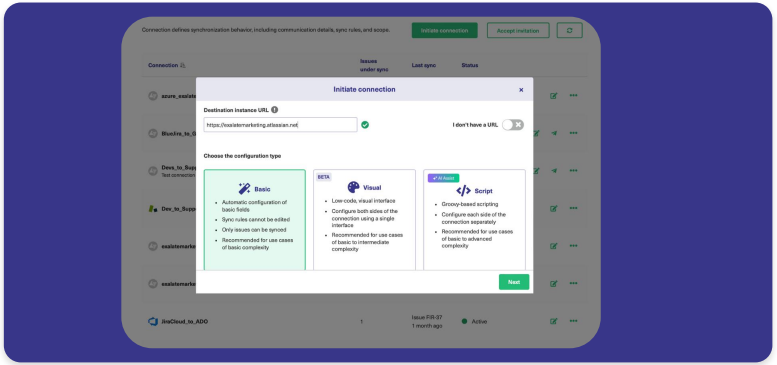
3.2.1 Configuration Modes

Configuration modes in Exalate Classic define how synchronization is customized between two instances.

To choose the configuration mode, first, the **EXALATE APP NEEDS TO BE INSTALLED ON BOTH SYSTEMS**. This ensures that the synchronization is decentralized and can be controlled from both ends. After this, a connection needs to be established between them. It is at this point that the destination instance URL, i.e. the link to the instance that needs to be synchronized with, is provided.

In this case, if the URL of the same instance is entered then the connection type is considered as local, otherwise, it becomes public. If the destination instance is on a private network or its URL is not known, then a unique invitation code is generated that can be copied at the destination instance.

Exalate suggests available configuration modes based on the destination instance version of Exalate.

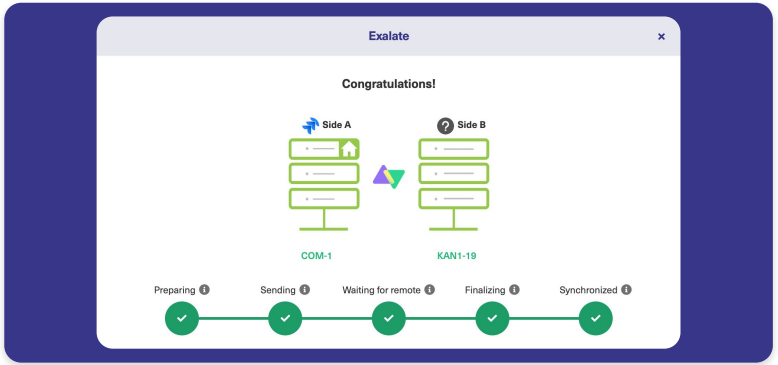


There are three types of configuration modes: Basic, Visual and Script.

NOTE

The Visual mode configuration type is in **BETA** version and only available for Jira (Cloud and on-premise), Zendesk and Azure DevOps.

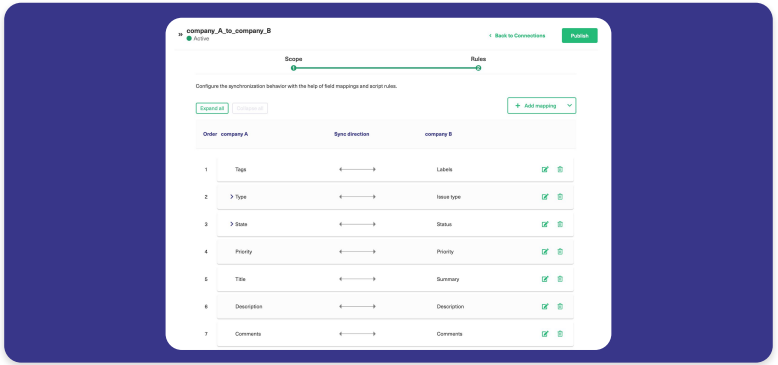
Basic Mode



This configuration mode allows for very simple synchronization configuration. For instance, in Jira, you can exchange the work item summary, description, attachments, comments, and issue types. It can be used for basic synchronization needs.

Changing the behavior of connections is not possible in the Basic configuration mode unless the configuration is upgraded to a Visual or Scripted connection.

Visual Mode



The **VISUAL MODE REQUIRES ADMIN ACCESS ON BOTH ENVIRONMENTS** to be able to implement the configuration.

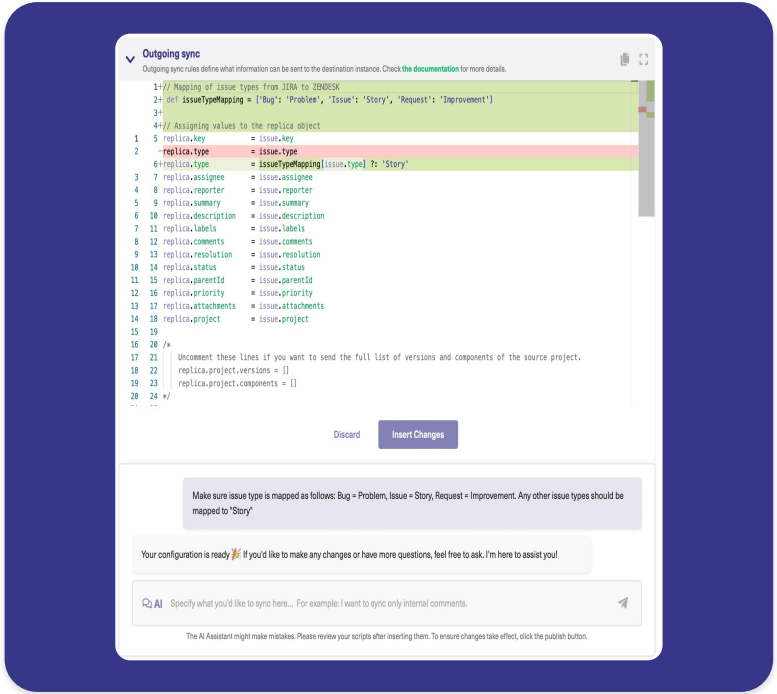
The Visual configuration mode (BETA) provides a no-code graphical user interface allowing for advanced mappings with a simple point-and-click interface. In this mode, one can add new mappings, edit or delete the existing ones, or simply drag and drop them to assign priorities. Advanced mappings can also be added with the help of scripts in this mode.

Script Mode

The Script mode is a low-code advanced configuration mode allowing to specify how the synchronization should behave in different situations. The flexibility provided through this mode is possible with the help of an integrated Groovy-based scripting engine. So one can define and model the sync rules in the most flexible way.

The Outgoing and Incoming sync on either side of the connection decide the information to be exchanged. The respective Outgoing and Incoming sync processors handle what needs to be sent and received by analyzing the scripts.

In Exalate Classic, the Script mode is enabled with AI Assist. AI Assist is Exalate's AI services that automatically generates outgoing or incoming integration scripts based on user defined prompts.



Accessing the Exalate Classic Console

The Exalate administrator is the authentication context that has access to the 'Exalate Classic Console'. With the Exalate Console, it is possible to configure and manage connections and the overall synchronization operation.

There are three types of Exalate administrators

3.2.2 The Local Exalate Admin

Local Exalate Admin access is required to configure and maintain all connections in all configuration modes from the local side. Such admin access is required due to the distributed architecture of Exalate. Since each side is autonomous, a local admin will ensure all connections and configurations are working as intended on the respective platform.

3.2.3 Admin on Both Systems (For the Visual Mode)

The Visual mode editor needs to have access to both sides of the connection to implement its functionality.

This admin access is required to configure a connection using the Visual configuration mode. It means that the user has administrative rights on **BOTH** systems.

As a first step in the configuration of the Visual mode connection, a verification process is started to ensure that the user has the correct access to both systems.

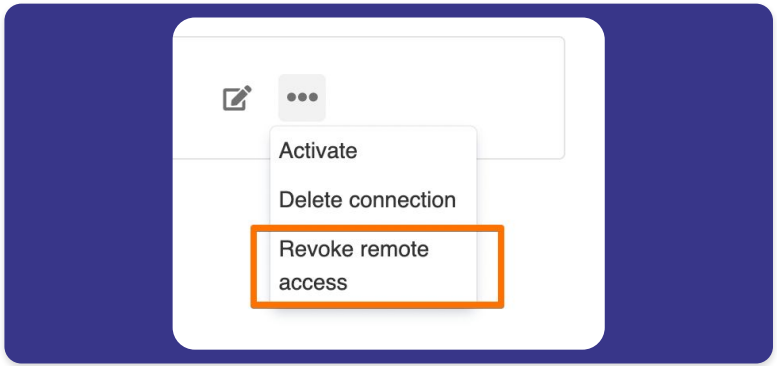
The way Exalate realizes this authentication when setting up a connection using Visual mode is through an implementation of the Oauth 2.0 protocol.

This means that system A will be granted access to system B on behalf of the admin user (which requires administrative access on both sides).

To do this, System A is required to register as an instance on System B. After registration, System A redirects the user to System B where s/he will prove that s/he has administrative access. Exalate will register the fact that this user has access to both systems.

Once authorization is given, System B redirects the user to System A, to complete the verification cycle. From that moment, System A will have been granted full Exalate admin access.

In case an admin user considers it necessary, he can always revoke access to an authorized instance.



3.2.4 The Project Level Admin

(Available currently since Sep 2021, only on Jira Cloud - version 5.2.0 and higher)

As a project administrator, it is possible to set up Basic mode connections without having system administrator rights.

As seen in section 2.4.1, the Basic configuration is fixed, and initiating/accepting such a connection is the only allowed configuration. Given this restriction, the configuration can be set up by a non-system administrator as well.

3.2.5 **Dedicated Admin Access:** Why Are Administrator-Level Permissions Required?

The Visual and Script configuration modes provide the possibility for deep integration with the underlying system through Groovy scripting. This flexibility also has the consequence that any data can be retrieved and modified. Users able to configure Exalate will have the capability to retrieve and/or modify any data. Access to the Exalate Console must be handled in the same way as access to the administration modules of the platform itself.

It is also therefore necessary that the authentication to the platform is handled by the platform itself. Exalate has no internal user directory.

NOTE

The Basic configuration mode has no configuration flexibility and can therefore be used by a larger audience.

3.2.6 Delegated Admin Access for Non-System Admins

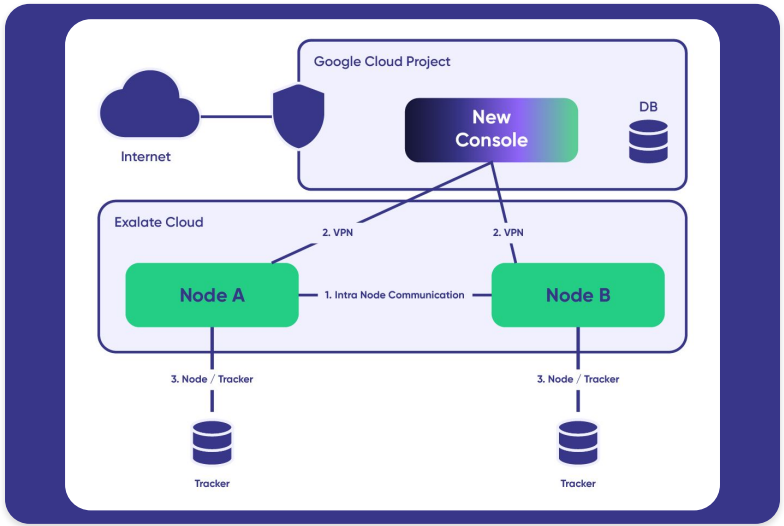
In some situations, you may need to grant non-system admins access to the Exalate Admin Console without giving away system-wide admin permissions. This would allow you to delegate admin maintenance tasks, such as adding, editing, and deleting Exalate configurations, create connections for all projects in Jira Cloud, as well as view and manage Errors, Sync Queue, Triggers, Clean-up, and Bulk Connect.

NOTE

More info about Delegated Admin Access for Non-System Admins can be found here

3.3 The New Exalate Console

3.3.1 Interaction Between The Internet, New Exalate, and the Nodes



New Exalate Console
Management interface hosted in Google Cloud in the St. Ghislain (Belgium). Provides configuration, monitoring, and administration capabilities. Connects to nodes via VPN for management only. The new exalate console database (DB) stores configuration metadata and connection definitions, not sync payload data.

DB (Console Database)
Stores connection definitions, console user details, and administrative metadata, including the activity history shown in the Activities tab. Does not contain synchronization payloads or customer data being synced between platforms.

Nodes (Node A, Node B)
Exalate nodes deployed per tracker instance within the Exalate Cloud infrastructure. Each node handles synchronization logic for its connected tracker, executing sync scripts and managing data transformation. Nodes communicate directly with each other for data exchange.

Trackers
The work management systems being synchronized (Jira, ServiceNow, Zendesk, Salesforce, etc.). Each tracker has a dedicated Exalate node that interfaces with it via APIs. Trackers can be hosted anywhere: on-premise, cloud, or hybrid.

Connection Types

#	Connection	Purpose
1	**Intra Node Communication**	Direct node-to-node data sync. All payload data flows here. Console excluded.
2	**VPN**	Console-to-node management channel. Configuration and monitoring only. No sync data.
3	**Node/Tracker**	API connection between each node and its tracker instance.

Key Point

The New Exalate Console is a management-only component. It connects to nodes via VPN for configuration and monitoring purposes, but **synchronization data flows directly between nodes** via the Intra Node Communication channel. The console never participates in, nor has visibility into the actual data being synchronized between trackers.

3.3.2 Accessing the New Exalate Console (Console Authentication)

The New Console (exalate.app) authenticates users through Keycloak, an open-source identity and access management solution deployed within Exalate's on-premise infrastructure.

Identity Provider Architecture

Keycloak as Identity Broker:

- Centralized authentication point for all new console access
- Standards-based protocols: OpenID Connect (OIDC) and OAuth 2.0
- Credential isolation from connected nodes: authentication occurs exclusively between user browser and Keycloak instance

NOTE

Node-level API access for data synchronization remains authenticated separately through existing node authorization mechanisms. The New Exalate Console's Keycloak authentication governs administrative access only: synchronization traffic between nodes does not flow through the Console.

Google Social Login:

- Users authenticate via existing Google Workspace or business Google accounts
- Keycloak acts as OIDC relying party to Google's identity provider
- Authentication flow: User → New Console → Keycloak → Google → Keycloak → New Console
- No Google credentials stored by Exalate; Keycloak receives only identity tokens containing email, name, and profile identifier
- Automatic account provisioning on first Google sign-in (if enabled by administrator)

Session Security

- Short-lived access tokens with refresh token rotation
- Sessions terminate on inactivity or explicit logout
- Token validation on each new 'new console' API request

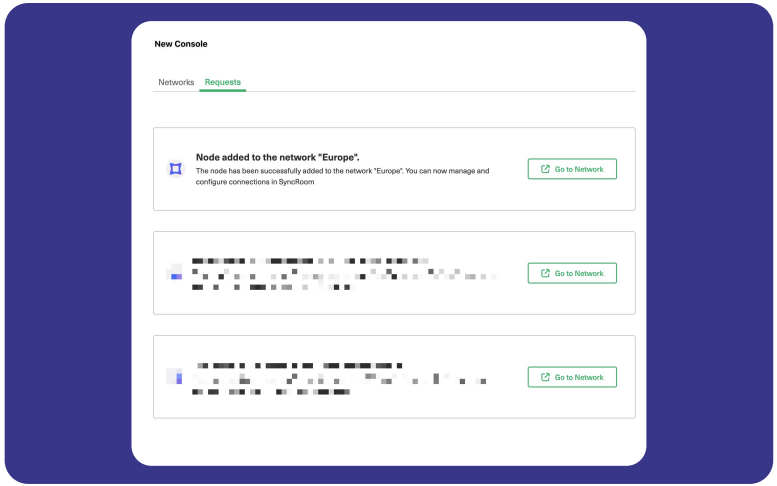
3.3.3 New Exalate Console to Node authentication

Node Authentication Security

The New Exalate console authenticates with Exalate nodes using JSON Web Tokens (JWT) signed with RSA-256 asymmetric encryption. A 2048-bit private key, held exclusively by the new console infrastructure, signs all authentication tokens; nodes verify signatures using the corresponding public key distributed during deployment. This asymmetric model ensures tokens cannot be forged, compromise of a node does not expose the signing key.

Authentication employs four validation layers: cryptographic signature verification confirms token authenticity; authorization token validation confirms prior administrative approval; instance targeting prevents cross-node token reuse; connection ownership verification ensures only the designated console instance manages each connection. Tokens expire after 10 minutes, limiting exposure window in case of interception.

Access establishment requires explicit administrator approval. The new console initiates an access request; a node administrator reviews and approves the request through the local Exalate interface; only then does the node issue the authorization token enabling ongoing API access. Administrators can revoke access at any time by removing the authorization record from the node's database. All communication occurs over TLS 1.3+ encrypted channels.



3.3.4 The New Exalate - dropping the need for elevated privileges

Exalate Classic requires proxy users to authenticate against the underlying tracker's permission system, typically necessitating elevated permissions (often administrator-level) to access the Exalate Console and manage sync configurations. This creates operational constraints: organizations must provision dedicated service accounts with broad access rights, security teams face audit concerns over administrative privileges, and permission scope frequently exceeds actual operational requirements.

New Exalate separates console authentication (handled by Keycloak) from proxy user permissions (configured per node), eliminating the dependency between console access rights and tracker-level permissions. Proxy users require no specific role or permission level; any user role with access to the target objects is sufficient. Permissions can be scoped to exactly what synchronization requires: read/write access to entities under sync, limited to specific projects, spaces, or objects.

Security benefits follow directly from this architectural change. Proxy users operate under the principle of least privilege, reducing attack surface if credentials are compromised.

Service accounts no longer require administrative privileges, simplifying compliance audits and aligning with standard access control policies rather than requiring security exceptions.

Console Authorization: Connection Roles
Keycloak establishes who a user is; connection roles determine what that user can do. Every user with access to a connection holds one of two roles:

Owner: full control of the connection, including updating system authentication, enabling or disabling the Sync Panel, activating, deactivating, and deleting the connection, and managing its users.

Collaborator: can work with scripts, triggers, and versions on either side, publish changes, and manage labels, but cannot update system authentication or perform connection-level administration.

Owners can invite Owners or Collaborators. Collaborators can invite only Collaborators and can remove only the Collaborators they invited. Every connection keeps at least one Owner, and the Owner who created the connection cannot be removed. Users removed from a connection are notified by email. A third role, **Sync Panel User,** has no access to the Exalate Console and is limited to the Sync Panel browser extension. Owners and Collaborators can both view the connection's change history in the Activities tab.

3.3.5 Node Provisioning and Proxy User Credential Management

During node provisioning (RampUp), the New Exalate console collects proxy user credentials required for nodes to authenticate with their respective issue tracking systems. The process follows a secure credential lifecycle from collection through transmission to permanent node storage.

Credential Collection

Administrators provide platform-specific authentication credentials through the New Exalate Console interface. Supported authentication methods vary by platform: Jira Cloud uses OAuth 2.0 via Atlassian Connect/Forge; ServiceNow supports OAuth 2.0; Azure DevOps Cloud supports Personal Access Token or OAuth 2.0 via Microsoft Entra ID (Azure DevOps Server: Personal Access Token only); Zendesk uses an OAuth 2.0 authorization flow; Salesforce uses Client ID and Client Secret from an External Client App; GitHub uses OAuth 2.0 via GitHub App installation on the organization or account; Freshdesk and Freshservice use an API key; Asana uses a Personal Access Token. The New Console detects the tracker type from the provided URL and dynamically renders the appropriate credential input fields. Sensitive fields (tokens, keys, secrets) display with client-side masking.

Delegate Authentication

The administrator creating a connection does not need credentials for both systems. Using Delegate Authentication, the creator can hand off authentication of either system, or both, by entering another person's email address instead of system details.

This supports scenarios such as a consultant setting up a connection between two organizations without direct access to either system. The creator never handles the delegated system's credentials.

The invitee receives an email invitation and signs in to the New Exalate Console, creating an account if needed. They are shown who invited them, which connection, and which system they are configuring, and they can authenticate only that system.

Their credentials then follow the same validation, transmission, and storage lifecycle described in this section. Until every delegated system is authenticated, the connection remains a draft and cannot be launched. Launch requires the creator to confirm a summary of both systems, their URLs, and their authentication methods. Invitees join the connection with the Owner role

Pre-Transmission Validation

Before credentials leave the new console, platform-specific authentication clients validate them against the target tracker's API. This step confirms credential validity, detects authentication errors before node setup, and prevents transmission of non-functional credentials. Validation failures return immediate feedback to administrators without proceeding to node configuration.

Secure Transmission

Validated credentials transmit to nodes via HTTPS/TLS 1.3+ with JWT-authenticated API calls.

Node-Side Storage

Upon receipt, nodes encrypt sensitive credential fields using Blowfish symmetric encryption with Base64 encoding before database persistence. Encrypted credentials persist in the node's PostgreSQL database, protected at rest.

Credential Cleanup

After successful transmission to the node, the new console purges temporary credential storage. This minimizes the credential retention window and reduces central attack surface: production credentials reside only on the nodes requiring them for tracker authentication.

3.3.6 Node Functionalities Used by the New Exalate Console

Node API Functionality Reference

The new console communicates with Exalate nodes to perform centralized management operations. The following capabilities are exposed through authenticated API calls from the new console to individual nodes.

Node Discovery and Information Retrieval

The new console queries nodes to retrieve metadata including node type, version, instance identifier, and current access status. This information populates the network topology view and validates node availability before management operations. The console also retrieves supported entity types (issues, epics, pull requests) and project lists from the underlying tracker to populate configuration forms.

Access Management

Before managing a node, the new console must obtain explicit authorization. Access requests are submitted to target nodes and require approval by local node administrators through the classic console interface. The new console polls access status until approval or denial, and can revoke access when removing nodes from networks.

Node Setup and Configuration

During provisioning (rampup), the new console transmits proxy user credentials and tracker URLs to nodes. This establishes the authentication context nodes require to communicate with their respective issue tracking systems. The console also validates connection name uniqueness before creation to prevent conflicts.

Connection Management

The New Console creates, activates, deactivates, and deletes connections on nodes. Both sides are registered (via ramp-up) or imported and approved as nodes beforehand; the Create Connection wizard then establishes the link directly between the already-authenticated nodes. If the creator lacks credentials for one or both systems, authentication can be delegated to another user; the connection stays in draft until every system is authenticated. Users can be added to a connection at any time from Connection Properties, with permissions governed by their role.

Trigger Management

Triggers define conditions under which entities enter synchronization. The new console creates, updates, and deletes triggers on nodes, supporting both query-based triggers (JQL, WIQL, ServiceNow queries) and project-scoped triggers. Trigger validation ensures query syntax correctness before activation.

Script Management

Synchronization behavior is defined through Groovy scripts executed on nodes. The new console retrieves, updates, and validates outgoing and incoming processor scripts. Script versioning enables rollback to previous configurations. The console also fetches IDE suggestions to provide autocomplete functionality in the script editor.

Test Run Operations

Before applying configuration changes to production connections, administrators can execute test runs. The new console initiates test synchronizations with specific entities, retrieves execution results, and displays script output and transformation previews. This validates script behavior without affecting live data.

Sync Queue Operations

The new console monitors and manages synchronization queues on nodes. Operations include retrieving queue items, resyncing failed transactions, clearing queues, and forcing synchronization of specific entities. Queue statistics provide visibility into sync backlog and throughput.

Entity Sync Status

Administrators can query the synchronization status of individual entities and retrieve lists of all entities currently under sync for a connection. This enables targeted troubleshooting and provides visibility into sync coverage.

Bulk Operations

For large-scale management, the new console supports bulk entity operations including mass resync and bulk exalate (bringing multiple entities under sync simultaneously). These operations accept entity key lists and execute against the node's tracker.

Cleanup Operations

When removing entities from synchronization, the new console can unsync individual entities or clear all synced entities for a connection. Cleanup operations remove sync metadata without deleting the underlying tracker entities.

Error Management

The new console retrieves, filters, and manages synchronization errors from nodes. Operations include resolving individual errors, bulk resolution by connection or selection, and ignoring non-critical errors. Error details include local and remote replica data for debugging script issues.

Support and Diagnostics

For troubleshooting, the new console can initiate support bundle generation on nodes, poll for completion status, and download the resulting diagnostic archives. Support bundles contain logs, configuration snapshots, and system information for support ticket resolution.

Sync Panel

The Exalate Sync Panel is a browser extension, available for Chrome and Edge, that brings sync status and actions to the items a user already works with. Inside the source system, the panel shows whether an item is under synchronization and, where available, a link to its counterpart on the remote side. From the panel, an authorized user can start a manual sync, connect an item to an existing remote entity, or unlink it, without leaving the system they are in. Access is not automatic: an Exalate Console administrator grants Sync Panel access per connection, so each connection's owner decides who can view status and act on items through the extension.

Access is not automatic: only a connection Owner can enable the Sync Panel and grant access, which is controlled per side and by email domain, so each connection’s Owners decide who can view status and act on items through the extension.

Dashboard

The Dashboard gives an administrator a single view of how many items are under synchronization across all of their connections, and how that number changes over time. It records a daily cumulative snapshot, so teams can follow trends and drill down from the whole network into any single connection. Consistent with the console's management-only role, the Dashboard reports how many entities are under sync, not what those entities contain. Because items under sync is also the basis for the subscription, the same view lets a team track its usage directly.

Activities

The Activities tab gives connection Owners and Collaborators a shared, read-only history of who changed what across their connections. It records connection setup, configuration and script changes, user management, sync actions, and subscription events. Events are kept for 90 days. Activity entries can be exported to a file directly from the Activities tab, so teams can keep records beyond the retention period or share them with auditors. Consistent with the console's management-only role, the Activities history records administrative actions, not the content of the entities being synchronized.

4. Exalate AI

Exalate's AI capabilities have evolved significantly with the introduction of Aida.

Previously, AI Assist helped generate context-aware scripts for implementing integrations based on the specific platform and existing sync configuration.

Aida represents a comprehensive evolution of these capabilities within the unified New Exalate console experience. In addition to script generation (previous AI Assist functionality), it now supports troubleshooting.

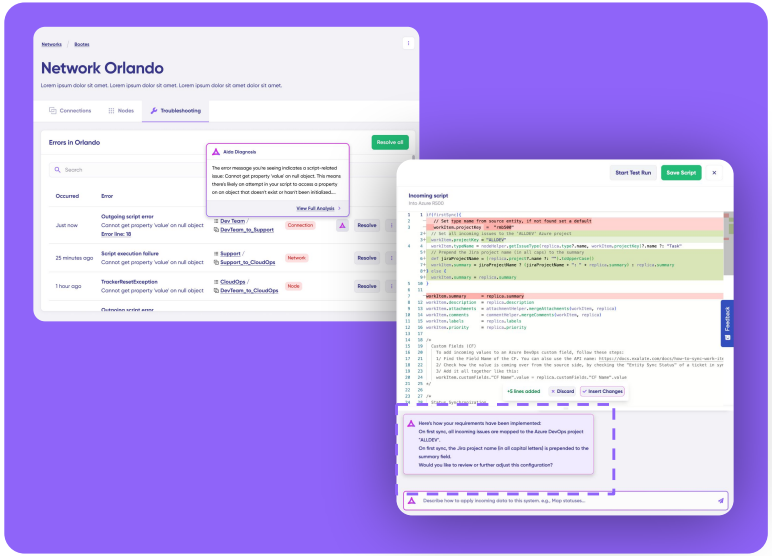
When errors occur, Aida explains them in plain language and provides context-aware suggestions for resolution.

NOTE

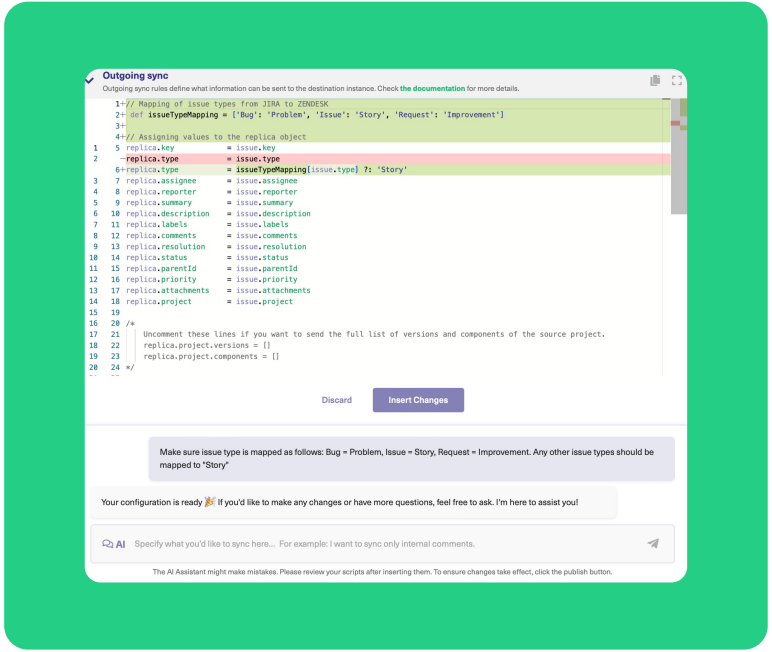
While AI Assist remains available in Exalate Classic, all new AI capabilities are available through Aida, which is designed to work with features like Test Run and script versioning.

4.1 Aida and AI Assist

Aida: New Exalate



AI Assist: Exalate Classic

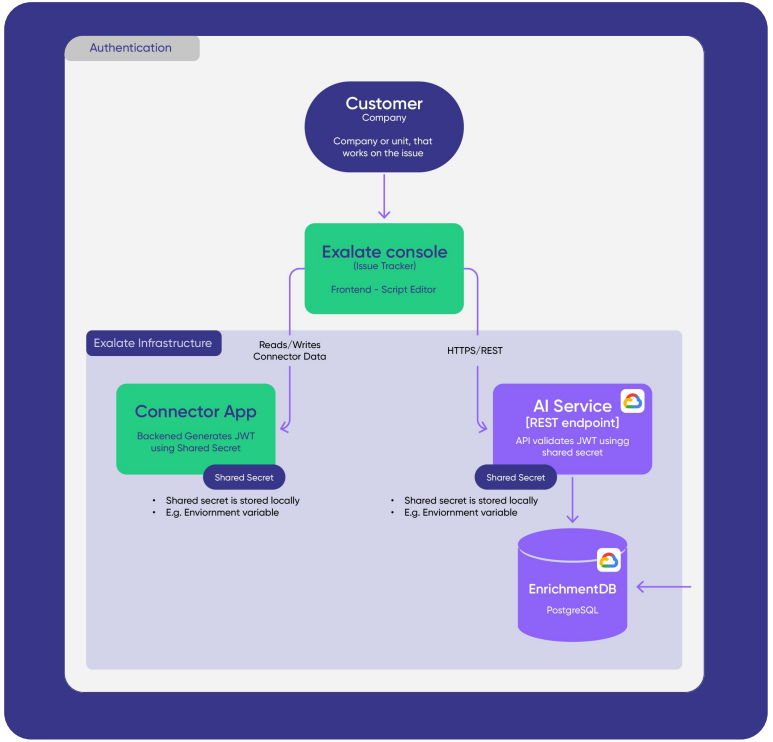


4.2. How are the AI Services Being Accessed

AI is a service within the Exalate Cloud architecture, designed to handle user prompts and generate scripts and triggers, or suggest changes to them.

4.2.1 System Diagram

The figure shows a visual representation of the system's architecture illustrating all involved components and the data flow.



4.2.2 Communication

- Protocol: All communications between the client and the AI Assist service are conducted over HTTPS, ensuring encryption in transit.
- Authentication: JSON Web Tokens (JWT) are utilized for authentication purposes. Each Exalate instance utilizes the JWT of the current node to authenticate requests.

4.2.3 Exalate AI: Risk Assessment and Mitigation

- **Data Breach**: The use of robust encryption (HTTPS) and authentication (JWT) minimizes the risk of unauthorized access.
- **Data Loss**: Regular backups mitigate the risks associated with data loss.

NOTE

If you have more questions about AI Assist refer to the FAQ section.

4.3 Exalate AI: Tracker access

Aida can access the tracker systems in scope of the user's integration. Aida only reads the system's structure metadata and not the data itself.

Aida has the same access control as other Exalate components.

Reference section 3.3.3

NOTE

If you have more questions about Aida refer to the FAQ section.

exalate

Secure, Real-Time Integration Across Tools

Connect Jira, ServiceNow, Azure DevOps, Salesforce, Zendesk, Freshdesk, Freshservice, Asana, and more. Bi-directionally and in real time.

- Total control over what data flows in and out.
- Built for internal teams and cross-company collaboration.
- Fully customizable to match your workflows.
- Maximum security, with proven certifications.

BOOK A DEMO | TRY FOR FREE | TRUST CENTER



5. Exalate Cloud

Exalate Cloud is a specialized Platform as a Service (PAAS) designed to host Exalate nodes.

The Exalate application connects to a single task management system and operates as a single-tenant application. This means it has its own dedicated process, storage, and database space. This design choice ensures the highest level of information security and minimizes the risks associated with vulnerabilities often found in multi-tenant applications.

Key Features:

- **Single-Tenant Application:** Each Exalate application has its own dedicated resources, ensuring maximum security.
- **PAAS:** Exalate Cloud is a platform optimized for hosting Exalate nodes, eliminating the challenges of on-premise deployment.
- **Kubernetes Clusters**: The cloud infrastructure is built on Kubernetes (K8s), an open-source system renowned for automating deployment, scaling, and management of containerized applications.
- **Datacenters**: Exalate Cloud nodes hosted in Antwerp run in Tier 4 datacenters, known for their fault tolerance and high redundancy, with a guaranteed uptime of 99.995% annually. Nodes hosted in other regions run on Google Cloud. See section 5.3 for all hosting locations.

Note:

More information about Exalate Cloud is available on trust.exalate.com.

5.1 Security Components

The whole configuration of Exalate Cloud is done through IaC principles (Infrastructure as Code) where every change in the environment is governed by the appropriate controls these environments require.

Exalate Cloud ensures a clean separation between the infrastructure to run a node and the node itself, enforcing the single tenancy of the solution. This means that there are no shared components in use.

- The database tables of a single node reside in a specific database scheme with its own user ID and password. The database server ensures that all the data is encrypted with a specific key for each scheme.
- The file storage is either created with its own partition, and all data is encrypted with a specific key for that partition, or as a separate S3 bucket with its own authentication
- The node itself is running as a container in a Kubernetes pod - such that it has its own process and memory space.
- The network traffic is encrypted from the node onwards using commercial-grade SSL Certificates.

The OS on the compute engines is based on hardened images patched at least once a month reducing potential attacks and security vulnerabilities. The patching of the OS is done in an automated manner without any human intervention.

The GKE version is based on the stable channel. Patches are applied within 2 weeks of release (after validation in staging). Security patches are automatically deployed through the Compute Engine auto-update capability.

Additionally, the Exalate cloud is protected with SentinelOne which guards the environment against multiple attack vectors. All incidents and vulnerabilities are reviewed and assessed SentinelOne on a 24×7 basis.

5.2 Access to The Environment

Access to the environment can only be done through service accounts which in turn are managed (and governed) through IaC.

Only staff with an appropriate security clearance have (logical) access to the Exalate Cloud through a jump box. Authorized staff have only a small subset of predefined commands that they can execute, and every command is logged.

Access to a specific node can only be done after the customer's explicit approval. No Exalate staff have physical access to the environment of Exalate Cloud.

5.3 Where Is Your Data Stored?

Find the data storage locations here.

Exalate Cloud nodes are hosted in one of the regions: EU, the US, Australia. The prompt logs for Aida and AI Assist are stored in Google Cloud in Belgium.

NOTE

The DPA provides more details on how data is handled and is available here.

6. Application Security

To limit the risks of security vulnerabilities in the application, a large array of security-related testing practices have been put in place.

6.1 Security By Design

Security is considered in every step of the development cycle.

- A security champion is appointed to maintain a threat model of the application and review all implementations (through code reviews) against that threat model.
- The security champion is involved in the refinement (requirements), preparation (specification), and validation of all development activities.
- Security vulnerability scanning is implemented by using the tools provided by Aikido that provide immediate feedback whenever a vulnerability is detected.
- Critical and blocking vulnerability-related fixes are handled as a top priority.

6.2 Atlassian Cloud Fortified

The whole configuration of Exalate Cloud is done through IaC principles (Infrastructure as Code) where every change in the environment is governed by the appropriate controls these environments require.

Exalate Cloud ensures a clean separation between the infrastructure to run a node and the node itself, enforcing the single tenancy of the solution. This means that there are no shared components in use.

Exalate is a participant in the Atlassian Cloud Fortified program which includes

- A Bug Bounty program managed through Bugcrowd.
- Continuous monitoring of core capability Service Level Objectives (SLOs) and tests.
- Agreements around incident and vulnerability management and customer communication.

NOTE

Note that these provisions also apply to the Exalate version for Non-Atlassian related products.

6.3 Penetration Testing

A yearly application penetration test is organized in January covering a wide array of vulnerability testing. From 2025 onwards, this cadence is increased to 3 times a year.

All pentest reports are available in our trust center.

6.4 Other Application Security Measures

The processors are a significant risk in the security model provided by Exalate.

The processors (incoming and outgoing sync processors) are executable code that can manipulate any information reachable by Exalate.

Therefore, it is vital that a governance model is put in place to ensure that the right people have access to the configuration interface - the Exalate Console.

- The scripts can be monitored for any change. In the new Exalate console, every script and configuration change is recorded in the connection's Activities tab, showing who made it, and can be exported for audit purposes.
- The core mapping functionality can be externalized to the file system and versioned through Git or other VCS systems.
- Access to the data is always performed as a proxy user. Therefore, the authorization of this proxy user needs to be governed by the underlying platform.

6.5 Security Best Practices: AI Assist and Aida AI

- **Secure Development Lifecycle (SDL):** Following SDL practices to identify and mitigate vulnerabilities during the development phase.
- **Encryption Protocols:** Implementing HTTPS for encrypted communication.
- **Identity Management:** Using JWT for secure authentication and user identity verification.
- **Data Protection:** Ensuring that data storage complies with relevant data residency requirements and that regular backups are in place.
- **Data Processing:** Data retrieved from tracker systems are always stripped from any credentials and PII data before being sent to any LLM Provider.
- **ISO 27001 Framework:** Security management and compliance are further reinforced by adherence to the ISO 27001 standard, which encompasses risk management, auditing, and incident response procedures.

Security FAQs

Check Security FAQs Here

Glossary

Active objects	Access to the database is controlled by the 'Active Objects' database layer provided by the Jira environment for Deployment type 2 of Exalate.
Active Version New Exalate	The configuration snapshot currently used by the Connection in production.
AI Assist Exalate Classic	It's an Exalate AI service available in Script mode. It auto-generates outgoing and incoming sync scripts based on user defined prompts.
An Exalate (verb) Exalate Classic	An Exalate is the action of bringing an entity 'under sync' - this typically results in creating an entity on the other side.
Archived Version New Exalate	A read-only configuration snapshot kept for history or rollback.
A Sync	A Sync is the action of bringing an item 'under sync' - this typically results in creating an item on the other side.
Atlassian Connect add-on	Atlassian Connect is a development framework for extending Atlassian Cloud products. Add-on apps can be developed using Connect in Jira Cloud.
Asana	Asana is a web and mobile work management platform designed to help teams organize, track, and manage their work and projects, connecting tasks to company-wide goals.
Azure DevOps	It is a software application providing version control, project management, automated builds, testing, and release management capabilities. It aims at narrowing the gap between Development and Operations teams.

Activities Tab New Exalate	A read-only history of changes made to a connection, such as configuration, script and user changes. Entries are kept for 90 days and can be exported.
Aida New Exalate	Exalate's AI service in New Exalate. It generates and changes sync scripts and triggers, and explains sync errors in plain language.
Basic Configuration Mode Exalate Classic	This is an Exalate mode that allows for default mappings between different platform fields. For instance, in Jira, you can synchronize Summary, Descriptions, Attachments, Comments, and Issue Types.
Blobs	A blob is another name for a 'file', or 'attachment' ...
Bug Bounty Program	Exalate is subscribed to the Bug Bounty program. Currently, around 200 researchers have contributed to identifying security vulnerabilities.

Bulk Connect Exalate Classic	This operation allows you to connect and sync entities from two different platforms in bulk.
Bulk Relink New Exalate	This operation allows you to link and sync existing items from 2 different platforms in bulk.
Bulk Exalate Exalate Classic	Bulk Exalate allows you to sync existing entities between two platforms that satisfy conditions you have set in Triggers in bulk.
Bulk Sync New Exalate	Bulk Sync allows you to sync existing items between two platforms that satisfy conditions you have set in Triggers in bulk.
Bulk Unlink New Exalate	Unlink all items that match a trigger query in a single operation.

CIA Triad	Confidentiality, Integrity, and Availability, form the cornerstone of an organization's security infrastructure.
Clean Up	This operations stops the entity synchronization.
Configuration modes Exalate Classic	Configuration modes in Exalate define how synchronization is customized between two instances.
Connection	A connection between two instances specifies the synchronization behavior. Exalate also verifies the source and destination instances through the connection between them.
Decentralized integration(Autonomy)	This Exalate feature ensures that both the integrating parties have control of what information is sent out and how incoming information is processed. This is the reason why the Exalate app has to be installed on both platforms.
Deployment models of Exalate	These specify how Exalate is being deployed: on-premise, as a cloud application, or as a hybrid approach. There are four deployment models supported.

Destination instance URL	In the context of a connection, the destination instance URL denotes the remote instance URL.
Draft version New Exalate	An editable configuration snapshot not yet active for live syncing.
Entity Exalate Classic	It is the information that needs to be synced. It is used to denote tickets, issues, incidents, cases, etc. depending on the platform on which it exists.
Entity URN Exalate Classic	Entity URN is a unique identification number that helps identify a particular entity on the different task management systems. This URN is used for syncing and unsyncing entities between different systems.
Exalate add-on Exalate Classic	In the deployment type-2, the Exalate add-on is used for the Jira Data Center.
Exalate application/ app	The bi-directional synchronization application that we discussed in this whitepaper.

Exalate Cloud	Exalate Cloud is Exalate's hosting infrastructure, optimized for hosting the Exalate nodes. It runs in Tier 4 datacenters in Belgium and on Google Cloud in the US, the EU, and in Australia. It provides security, high availability, scalability, and performance.
Exalate Console	The Exalate Console is used to configure and manage connections and the overall synchronization operation of a single node.
Exalate Network	An Exalate network is an ecosystem of multiple connected trackers. For instance, a car parts manufacturer is using Exalate to integrate their work management system with all the work management systems of their customers.
Exalate Node/ Node	An Exalate node is an instance of the Exalate application

Exalate Post function	This function is available in Jira to start synchronization automatically.
Exalate Proxy User	A system user used by Exalate to access the entity information. This account is used to make changes, such as creating issues and updating data
Exalate REST API	Access to Exalate functionality, such as transport and console, is controlled by the Exalate REST API.
Exalate Sync Panel	The Sync panel is used to denote the status of the entity under synchronization and also provides a link to the remote entity if available.

Exalate transport layer	Exalate is a message bus that exchanges structured messages between multiple platforms. As a software application, Exalate has a layered architecture, where the transport layer is responsible for transmitting data between the nodes.
Force Unlink New Exalate	This operations break the link between items, stop future syncs and clears any queued transactions.
Full Exalate admin Exalate Classic	A single user having access to both platforms under synchronization to configure a connection in the Visual mode.
Freshdesk	Freshdesk is a user-friendly help desk that is easy to set up and has AI-powered features. It is commonly used by customer service or support teams.
Freshservice	Freshservice is an ITSM solution designed to streamline and improve IT service delivery. It offers features like incident, change, problem, and asset management.

GitHub	GitHub is an open-source software development and version control application.
Governance model	A Governance model is suggested to make sure unauthorized access to the Exalate Console is avoided and proper permission schemes are in place.
Groovy-based scripting engine	It is an advanced scripting engine that interprets and processes scripts written in Groovy for transforming and filtering the information to be exchanged.
HTTPS	A secure transfer protocol used by Exalate.

Integrated retry mechanism	Exalate's in-built mechanism to recover from system failures and downtimes without manual intervention.
Invitation code Exalate Classic	A unique code that authenticates the source and destination instance to form a connection. It is used as a digital signature to verify both parties. Consider it as a key, and make sure that nobody has unauthorized access!
Instance	An Instance is a task management system with the entities that you want to synchronize.
Integration	A container that links two systems and groups all Connections between them.
Items New Exalate	Item that you want to synchronize with the destination system. They can be issues in Jira and GitHub; incidents, problems, change requests in ServiceNow; Cases, Opportunities in Salesforce; work items in Azure DevOps or tickets in Zendesk.

Item ID New Exalate	Item ID is a unique identification number that helps identify a particular item on the different task management systems. This ID is used for syncing, linking and unlinking items between different systems.
Item Sync Monitor New Exalate	The UI section where you search, inspect, and operate on item-level syncs.
Jira	Jira is an issue or bug-tracking system developed by Atlassian. It also allows for agile project management.

Jira Service Management	Atlassian's Jira provides service desk capabilities for raising customer bugs and providing support.
Jira Software	Jira Software is part of a family of products designed by Atlassian to help teams of all types manage work. It is a bug or issue tracker and project management software.
JWT token mechanism	A JWT token mechanism is used to sign every message between the Exalate nodes such that the integrating parties are recognized as a trusted source.
Link (verb) New Exalate	Establish a synchronization between two existing items across two systems.
Local connection type Exalate Classic	A local connection allows for synchronization between local entities within a single platform. A typical example is a local connection between a Jira Service Management project and a Jira Software project.

Local Exalate admin	Local Exalate admin configures and maintains all connections in all configuration modes from the local side.
Loosely coupled	Exalate supports a distributed architecture where each side controls what is sent and received. This makes the systems loosely coupled and less dependent on one another, reducing maintenance challenges.
Middleware	Exalate acts as a middleware that transports data between two instances.
Multi-tenant	A multi-tenant architecture is an architecture where a single application provides support for different subscribers. The shared infrastructure is a potential security risk and needs to be continuously validated against information leaks.

Outgoing and Incoming sync processors	These processors are used to configure, filter, and determine what information is sent and received between the two instances.
Outgoing sync event	When the outgoing sync processor sends out information, it gets registered as an outgoing sync event.
Platform	A platform denotes various applications that Exalate supports, for instance, Jira, GitHub, Salesforce, Azure DevOps, Zendesk, etc.
Polling/Poll	Exalate periodically polls the instances to check for any synchronization updates to be applied to the destination side. This polling approach is applied where the push model is not supported by the platform.

Private to Public connection type	A connection type where information needs to be synchronized between the public-facing server and an internal server behind a firewall. No need to open a TCP port on the firewall to allow bidirectional synchronization.
Project level admin	Project level admins can set up Basic configuration-based connections without having system administrator rights.
Public to Public connection type	Z Connection type when information needs to be synchronized between two public-facing servers.
Push function of Exalate	Exalate registers the instance to receive webhooks whenever the tracker provides this capability. Used in deployment type 4.

Reverse proxy function of Exalate	All Exalate nodes are fronted with a reverse proxy for SSL termination.
Salesforce	It is a customer relationship management platform that enhances the customer experience and can be used by a variety of teams like sales, marketing, IT, and the like.
Scripts	Scripts are low code snippets written in Groovy scripting language to configure what information is sent and received.
Script/ Scripted/ Scripting mode	The Script mode is used to configure connections using scripts allowing for extended flexibility.

Security FAQ	A document that can be requested by email to security@exalate.com that can help answer commonly raised security queries.
ServiceNow	An ITSM platform that helps companies manage digital workflows for enterprise operations.
Single tenant setup (Exalate)	The environment provided by Exalate is such that the process, file, and database space for one customer is completely separated from the other.
Sync pair New Exalate	A pair of items that are related to each other through a synchronization relation. Any modification is synchronized towards the other item using the specifications of the relation.
Sync queue	A queue to register outgoing and incoming sync requests. Can be useful in case of downtimes, to apply the lost changes in the same order as their initiation.

Sync rules	Sync rules are groovy-based scripts to implement filtering, mapping, and transformation that can be applied to the information being shared between two instances.
Synchronization process	A detailed, step-by-step procedure on how the transactional synchronization engine works.
System New Exalate	A System is a work management system with the items that you want to synchronize.
Test Run New Exalate	A safe, non-destructive simulation of a sync to preview which items would sync and how.
The admin on both systems	A user who has admin rights on both platforms.
The local exalate admin	A user who maintains and configures the connections on the local instance.

The Mapper	The mapper is a reverse proxy that will redirect calls from the Jira Cloud to the right node application using the ID provided by Atlassian.

To trigger an Exalate	To trigger an Exalate means to bring an issue under sync. Exalate is a word play on the term ‘to escalate’.
Transactional synchronization engine	Exalate has an advanced transaction-based synchronization engine where every change is recorded and queued such that it is applied in the correct order.
Trigger	Start synchronization of issues or other entities automatically with the help of triggers.
Twin Exalate Classic	A Twin is a pair of entities that are related to each other through a synchronization relation. Any modification is synchronized towards the other entity using the specifications of the relation.
Unexalate Exalate Classic	The Unexalate operation is used to stop the entity synchronization. It is available as an option in the Sync panel on certain platforms and on the Entity Sync status tab in the Exalate Admin console.

Under sync	An entity (or issue/ticket) is under sync when it is connected to a remote issue with which it is synced.
Unlink New Exalate	The Unlink operation is used to stop the item synchronization. It is available as an option on the Item Sync Monitor section of Exalate.

Version New Exalate	A saved snapshot of a Connection's configuration at a point in time.
Visual mode Exalate Classic	A low-code, point-and-click user interface. Also has the option of adding scripts through advanced logic for enhanced flexibility. This helps accommodate unique synchronization needs.
Visual mode admin verification process	Verification process where a user is checked for credentials to the destination instance. Upon verification, the Visual mode connection is successfully established.
Work management system	Systems that are used to manage day-to-day business work. Examples are Jira, Azure DevOps, etc.

X-Exalate-JWT	Every request from the client has a token "X-Exalate-JWT" - allowing it to validate the message.
Zendesk	Zendesk is a service desk to serve customer tickets and other customer communications.

IF YOU HAVE ANY DOUBT/ QUESTIONS/ FEEDBACK ABOUT EXALATE OR THIS DOCUMENT, PLEASE SEND US AN EMAIL AT SECURITY@EXALATE.COM

exalate

The Leading Cross-Company Integration Solution

Exalate is the leading and the only solution specifically built for intra and cross-company integration scenarios with the most complex synchronization use cases.



The first thing that hits you when using this app is how easy it is to get a synchronization up and running. By default, everything works so well.

With the amazing support team and the flexible capability of Exalate, we were able to see it work nicely and become an important part of our work life cycle.

Exalate is marketed as the most flexible synchronization tool for issue trackers and that short definitions is pretty accurate from what we have observed.

BOOK A DEMO

TRY FOR FREE

exalate

Let’s get your integration up and running

Creating an integration that really suits your business needs has never been faster. Start at your own pace or or book a live walkthrough with our team today.

BOOK A DEMO

TRY FOR FREE

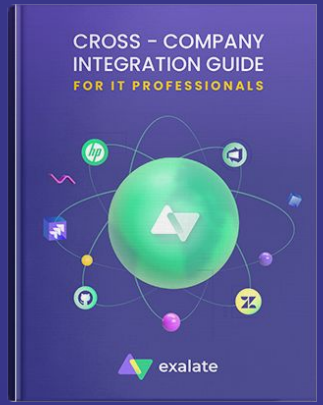


The first thing that hits you when using this app is how easy it is to get a synchronization up and running. By default, everything works so well.

With the amazing support team and the flexible capability of Exalate, we were able to see it work nicely and become an important part of our work life cycle.

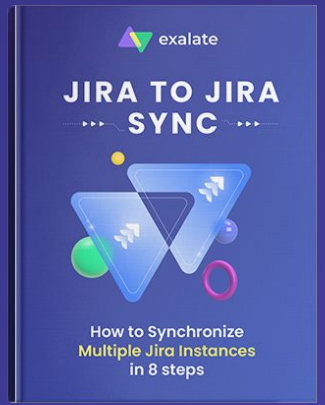
Exalate is marketed as the most flexible synchronization tool for issue trackers and that short definitions is pretty accurate from what we have observed.

Learn more



The Definitive Guide To Cross-Company Integrations for IT Profes

DOWNLOAD THE EBOOK



The Complete Guide to Synchronizing Multiple Jira Instances

DOWNLOAD THE EBOOK



The Project Management Blueprint for Agile Collaboration Between Teams

DOWNLOAD THE EBOOK