Guide to Contributing to Jakarta EE

Ways of Contributing

There are many ways to contribute to Jakarta EE 12, depending on your time, skills, and interests. You can start small and gradually get more involved as you become familiar with the community.

Stay Informed and Join the Discussion

A simple first step is following a Jakarta EE technology that interests you. The easiest way to do this is by subscribing to its mailing list. All Jakarta EE mailing lists are available here, and below, we highlight key lists for some of the most relevant Jakarta EE 12 technologies. Feel free to join discussions—your perspective as an end-user is incredibly valuable.

Advocating for Features

If you have a specific feature you'd like to advocate for, start by discussing it on the mailing list. If the feature isn't already tracked, you may need to open an issue detailing your proposal. Each Jakarta EE project has a GitHub repository with an issue tracker, which you can explore here. We've also linked key issue trackers, issues, and relevant discussions below.

Contributing Code, Documentation, and More

If you want to help implement a change, check the issue trackers for open tasks and ask to take one on via the mailing list. You can contribute in several ways:

- Enhancing APIs
- Improving the Technology Compatibility Kit (TCK)
- Updating documentation
- Contributing to technology implementations like GlassFish, Jersey, Mojarra, OpenMQ or EclipseLink

Providing a proof-of-concept, if possible, is a great way to advocate for change, though it's not expected from everyone.

Not Sure Where to Start?

Just join a mailing list, introduce yourself, and say you'd like to help. A Jakarta EE Ambassador should be available to guide you. You can also connect with the community through the Jakarta EE Ambassadors Google Group.

Eclipse Foundation Paperwork

There is no need to fill out any paperwork just to follow a project, record issues, and participate in discussions. You will only need paperwork for anything beyond that like contributing code or documentation.

When you are ready to take that next step, please make sure to sign the Eclipse Contributor Agreement (ECA). All you need in order to be a code contributor is to sign the ECA, that's it. You can complete it as an individual without any signatures from your employer. You don't need to pay anything. You may need some paperwork from your employer if you want to be a committer instead of a contributor. Being a committer typically comes much later. It's something you don't need to worry about right away.

If you have any questions about this, please let us know. For further reference, feel free to read this blog post by Wayne Beaton, director of open source projects at the Eclipse Foundation. Most of it is focused on the process of becoming a committer. There is also a FAQ on the ECA.

Getting involved in Jakarta EE 12 is a great way to make an impact, whether you're sharing your insights, advocating for new features, or contributing code. Join the conversation and start shaping the future of Jakarta EE!

Jakarta EE 12 Contents

There is not yet a formal roadmap for Jakarta EE 12, and you don't have to wait for one to begin contributing. The general expectation is that Jakarta EE 12 will be delivered some time in the next two years. You can begin work to move forward Jakarta EE 12 right now.

Ultimately each project decides which changes are implemented, and you can help influence that yourself. You should begin engaging with whichever technologies and features interest you the most. It is likely only a subset of technologies will change for Jakarta EE 12. Below are some key changes we feel pretty confident about (slightly more uncertain changes have question marks). By no means should you treat these as a sure thing or the only sensible changes. These are relatively well-vetted key ideas in the Jakarta EE Ambassadors community (most of these ideas are relatively long-standing and well understood).

So far, the envisioned changes more or less follow these general themes:

- Aligning specifications and features to take better advantage of Contexts and Dependency Injection (CDI), making CDI truly the central component model for Jakarta EE. This includes providing similar capabilities to EJB as a technology in favor of equivalent, modernized, more flexible CDI-centric functionality.
- Achieving greater portability and vendor-neutrality by standardizing more commonly used features and technologies that are vendor-specific today or available outside

Jakarta EE. Some of these represent long-standing feature gaps in existing specifications. Some entail adding new specifications for areas such as configuration.
- Aligning specifications and features to take better advantage of Java SE innovations. This includes making more technologies usable standalone, outside of a Jakarta EE platform implementation and having more standalone TCKs. It is likely the Java SE version supported by the platform will be upgraded to Java SE 21/Java SE 25.

Platform Level Changes

Below are possible Jakarta EE platform-level changes. While it is best that the changes are agreed upon at the platform level, it is possible to go ahead with the work with each Jakarta EE technology. There are several mailing lists where platform-level changes can be discussed, but the best one to start with is likely the Jakarta EE specification mailing list. There is a separate mailing list for TCK discussions.

- Adopt HTTP/3, at least in Servlet to start with.
- Adopt Records across the platform including in JSON Binding.
- Remove Java SE Security Manager dependencies across the platform.
- Using CDI across the platform wherever possible such as adopting CDI/**`@Inject`** instead of **`@Context`** injection in Jakarta REST.
- Deprecate the Application Client Container (ACC).
- Help clean up and prioritize issues.
- Modernize TCKs to utilize JUnit better. While a lot of work has been done in Jakarta EE 11 to have completely separated, modernized TCKs, there is more work to be done. In a similar vein, there is work left to adopt Maven as the build system across all Jakarta projects.
- Introduce a **`@Service`** CDI stereotype that seeks to provide similar capabilities to EJB **`@Stateless`**.

Individual specifications

Security

Jakarta Security already brought an important set of changes in Jakarta EE 10 and 11. However, there were a number of important pending changes that could not be implemented because of time. Security is also an important vehicle for providing CDI-friendly equivalents of EJB functionality. The best place to start discussing these changes is the Jakarta Security mailing list.

1. Addressing JWT/OAuth support (especially for use with OpenID Connect), including handling multiple authentication/authorization mechanisms in the same application.
2. Providing CDI-friendly, modernized equivalent for **`@RolesAllowed`**.

3. Adding an EL-enabled authorization annotation in addition to **`@RolesAllowed`** (e.g. **`@Authorized(“callerPrincipal.name == ‘Arjan’”)`**).

Concurrency

In Jakarta EE 10 and 11, Jakarta Concurrency introduced a number of long-pending changes to improve portability, usability and vendor neutrality. Largely due to timing, a few important changes were missed. Concurrency is also an important vehicle for providing CDI-friendly equivalents for functionality available only in EJB, often supplied in vendor-specific ways. Many of these changes have already been discussed in the EJB and CDI projects. The best place to start discussing these changes is the Jakarta Concurrency mailing list.

- Adding CDI-friendly equivalents for **`@Schedule`** and **`@Lock`**.
- Adding a **`@MaxConcurrency`** annotation.

Messaging

Jakarta Messaging is an important vehicle for providing CDI-friendly equivalents for functionality currently available only through Message Driven Beans (MDB). These are changes that have been extensively discussed in the EJB and JMS projects, but not implemented due to resource constraints. There are also other features that may help make Jakarta Messaging more compelling to a broader set of developers. The best place to start discussing these changes is the Jakarta Messaging mailing list.

- Provide CDI-friendly equivalents for MDB.
- Provide a Java SE/standalone bootstrap API.
- Introducing Messaging Lite geared towards cloud native use cases.
- Include AMQP interoperability?

New APIs

Aside from changes to existing APIs, a few new APIs could be added as part of the Jakarta EE umbrella.

Config

Jakarta Config is a specification that aims to address the ability to externalize configuration. This has been a long-standing gap discussed since Java EE 6. A key goal is that specifications that need significant configuration can consume it from outside the application such as the environment. Some example specifications that could use this functionality include Persistence, NoSQL, Messaging, and Mail. The best way to get involved is to explore the project, including joining the mailing list.

NoSQL

Jakarta NoSQL is a specification that aims to enable NoSQL access for Jakarta EE applications. It could be added as part of the Jakarta EE umbrella. The best way to get involved is to explore the project, including joining the mailing list.

MVC

Jakarta MVC is a specification that aims to provide a more action-oriented Java web framework for Jakarta EE applications. It could be added as part of the Jakarta EE umbrella. The best way to get involved is to explore the project, including joining the mailing list.

RPC

Jakarta RPC is a new specification that aims to bring gRPC support to the platform. The concept is very similar to Jakarta REST. It could be added as part of the Jakarta EE umbrella. The best way to get involved is to explore the project, including joining the mailing list.

Post Jakarta EE 12

There are a few further changes on the horizon that are likely too early for Jakarta EE 12. Many of these features could use further maturing, analysis, development, and experimentation – perhaps outside Jakarta EE first. You should feel free to engage the broader community in discussion around these changes.

- JSON schema support in JSON Processing and JSON Binding.
- Jakarta Mail is a relatively mature and feature-complete technology. There is, however, possibly an important usability gap to address: adding a higher-level CDI-based API that would provide a better abstraction for modeling common, non-trivial use cases such as sending a complex email with HTML or attachments. This is similar to the JMSContext API included in Jakarta Messaging. The likely best approach to this change is implementing an experimental open-source CDI extension within a project like Apache DeltaSpike.
- Jakarta Transactions is a mature API that remains the sensible choice for managed transactions in Jakarta EE applications. However, an important gap that could be filled is further optimizations that are relatively commonplace such as last resource commit optimization/one-phase commit/local transactions (e.g. WebLogic specific features that could be requested to be contributed by the vendor to the specification as basis for further improvement). The best place to start discussing these changes is the Jakarta Transactions mailing list.
- Addressing testing in a generic, holistic fashion for Jakarta EE applications. This work could be a specification or simply an EE4J umbrella project. Arquillian, MicroShed, TestContainers, and JUnit 5 are all possible technologies to leverage or get inspiration from.