Beyond Jakarta EE 11

Objective: Generate excitement about the future of Jakarta EE with ideas, messaging, and content showcasing where we expect the community to take it beyond the next release.

Method: Brainstorm about directions we should take Jakarta EE after release 11 is out. This will involve working sessions with the Steering Committee, Spec Committee, Marketing Committee and member organizations. Generate a working document that will influence the next generation of specs, messaging, content, and more. Integrate top ideas into the next developer survey to gather feedback.

Recommendation: Form an interest group (Jakarta EE Next ?) to prioritize and explore the topics in greater detail, especially those beyond the typical Jakarta EE scope.

Topics of interest:

Quantum Computing	Can we improve the ways Java developers leverage Quantum as part of their applications? ● Article - Quantum Computing for Java Developers | by Johan Vos | Medium
AI/ML	Are there APIs and patterns for use available for integrating AI and ML into Java applications? ● Potential to create a subgroup to explore opportunities ● APIs to make it easier to connect to data management and verification techniques/technologies ● APIs to connect private LLM ● Codify best practices and patterns that will influence the data used by generative AI tools when building Java applications.
Robotics	Are there areas of Robotics that would benefit from standardization and Java language support beyond what is currently available through APIs? We should consider the cloud impact as well.
Edge Computing	Move computing closer to the edge - distributed resources from IoT to cloud. Explore real time applications (LTI, lower latency) Add application in healthcare (E.g. Emergency buttons) - MQTT protocol to JMS or MQTT spec - Determine if we should come with our own solution/API to allow direct

	communication, or if we use a prepared solution
Green Platform / efficiency is king	With the Microservices hype being superseded by AI it's possible to talk about architecture again. From a green IT stand point every server shut down is a good server. The Java platform itself has excellent support for figuring out where all your resources go (CPU / RAM etc). However Jakarta EE does not leverage this up to now. This goes way beyond what general metrics and distributed tracing provide. The message would be that you do not need to scale out or scale up if you run on Jakarta EE. That is because the resources are managed in a very efficient way and the platform supports you on every step of your way to figure out how much of and on what your resources are spent. E.g. there could be a flame graph export for every type of request, for each large batch run and so on. No more guessing why your application became slow and how to change it.

Specification usage and ease of use:

Supersede EJB	EJB is a huge marketing impediment for the platform as a whole. If we do not definitively supplant it soon, it will be difficult to change perceptions no matter what else we do. These are the things we need to do, roughly in order. ● Define how to inject and use EntityManager in a thread safe way outside of EJB in plain CDI beans, especially using the @Transactional annotation. ● Providing CDI-friendly, modernized equivalents for @RolesAllowed and @RunAs. ● Adding CDI-friendly equivalents for @Schedule and @Lock. ● Adding a @MaxConcurrency annotation. ● Introduce a @Service CDI stereotype that seeks to provide similar capabilities to EJB @Stateless. ● Provide CDI-friendly equivalents for MDB.
Modernize Messaging	A lot of people still associate messaging in the Jakarta EE ecosystem to Jakarta Messaging, IBM MQ, ActiveMQ, etc. If we feel we should have a firmer footprint in the messaging space, we really should consider a Messaging Lite. This would be very important from a marketing/perception perspective. It would essentially be a modernized subset of Messaging geared towards cloud native use cases. The actual work may not be that much to get this done. Along these lines, it would be good to

	provide a Java SE/standalone bootstrap API for Messaging. ALso explore integration with popular messaging platforms, such as Teams, Slack, etc.
Modularity	How do we make it easier to use specs in different combinations? - Enable users to select individual specifications and have any dependencies automatically included. More profiles doesn't address this issue.
Spec support	1. Explore the reasons why certain specs are not getting support from the committees to move forward 2. Determine what specs are getting used, which are not, and why. 3. Provide an easy query form for developers to know which specs are appropriate for particular tasks 4. Provide a way to visualize the specs and their dependencies 5. Look at developing scenarios and determine which specs support those. How can we surface the information in a more client-centric way and/or a statement-base that would better support generative AI 6. Documentation and tutorials that are clear, concise and use a consistent approach that can be easily parsed by ChatGPT+others.
Industry specific API Patterns	1. Look to identify common spec usage across different industries and how we can package and promote to others in the industry 2. Kubernetes specific spec?

Other ideas:

Better defaults for Persistence	Make persistence.xml optional/an empty marker in Jakarta EE applications. For many applications, this is a perfectly fine starting point. If there is a data source and Persistence resources, just enable it easily.
Less XML in Batch	Add a Java batch job definition API as an alternative to XML.
Pluggable caching in Persistence	Support JCache as a second-level cache provider for Persistence.

Modern packaging	Make bootable/fat/executable jars mandatory for Core Profile. Most modern implementations do this already anyway. It's an easy marketing win.
Incubator	Explore an incubator process for new ideas aligned with Jakarta EE and Java usage.
	Translation to multiple languages to ensure better data available across language groups.
Partnerships	Look at partnering with other Open Source communities that focus on lifecycle issues beyond the application coding - build, deploy, monitoring - eg CNCF
Spring Analysis	We should review what Spring has - review obvious gaps and QA. Compare advantages of Jakarta vs Spring from a marketing perspective to improve perspectives and elevate Jakarta EE.
Excluded from EE11	Jakarta MVC, Jakarta Config, Jakarta no-SQL, Jakarta RPC, etc.
Normativity of API Jars and module-info and package-info.java	Currently none of these things are normative. This is a blessing and a curse. Should we re-evaluate this stance?
Documenting the "JDK N and N-1" pattern.	Consider what happened in EE 10 and EE 11 regarding JDK version. In both cases we ended up stating that the EE platform needed to have a Ratifying Compatible Implementation on the latest LTS JDK at the time **and** the previous LTS at the time. (EE 10: 11/17; EE 11: 17/21). I think we should accept this pattern **with one important exception**: If a component spec is dead set on introducing a binary dependency on the latest JDK, we do want to allow this.