Java EE Survey Results and Java EE 8

December, 2016

Java EE Development Team, Oracle

Table of Contents

Abstract/Summary

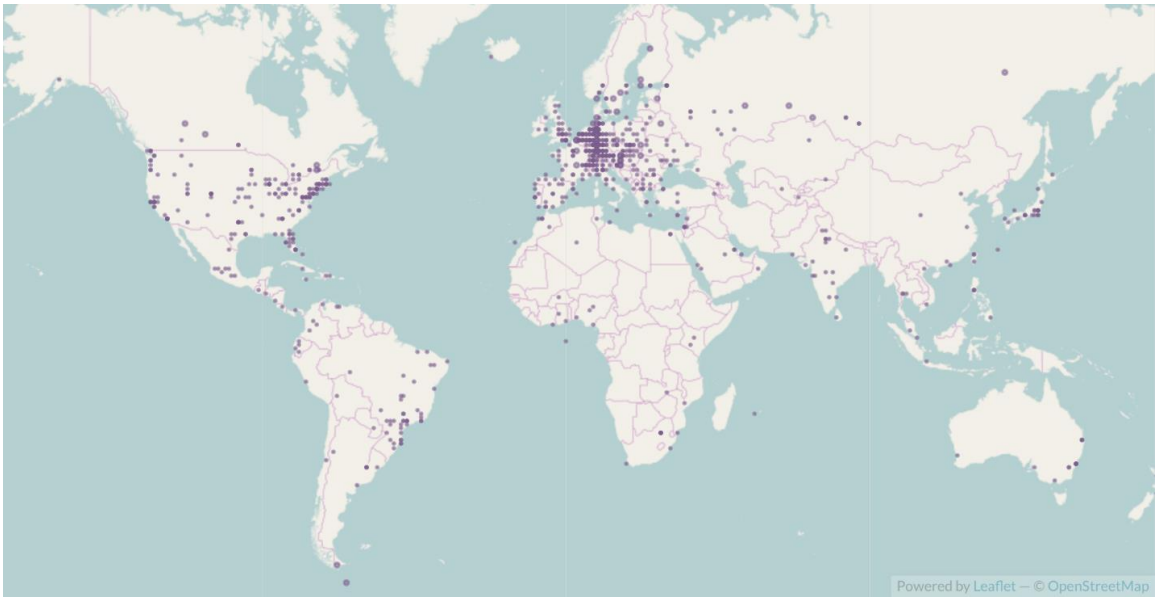
This document describes the results of the Java EE survey conducted in September-October 2016, concerning future enhancements to Java EE. In particular it describes the community ranking of the importance of future Java EE component technologies. The document also summarizes the conclusions for our Java EE 8 proposal, based on survey results and additional review of implementation considerations.

Survey Background

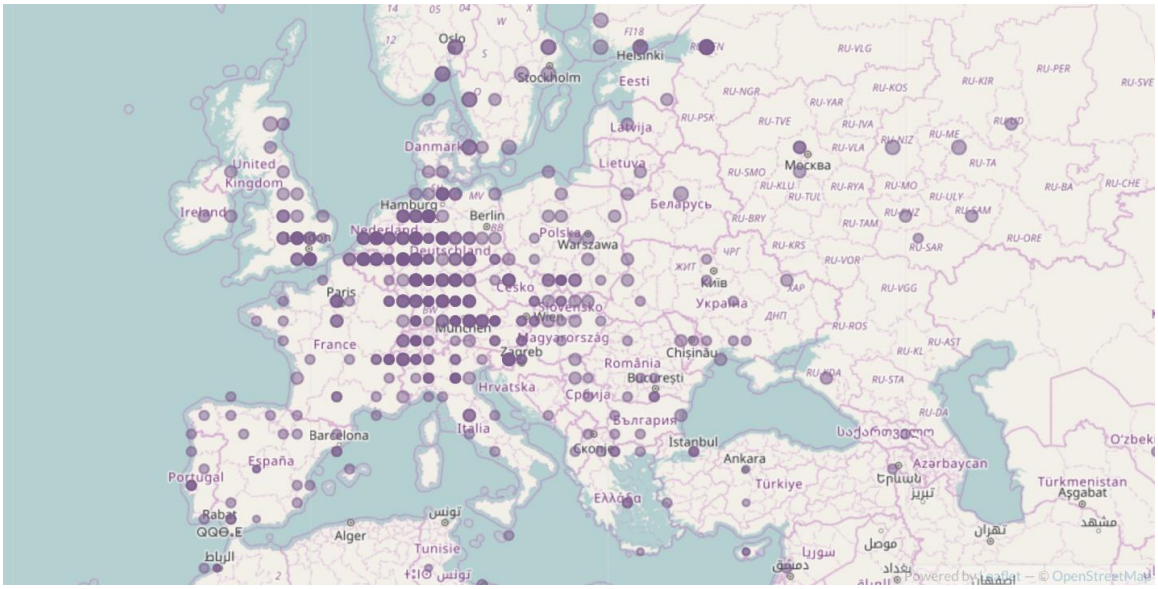
We conducted this survey in the context of its revised Java EE roadmap presented at JavaOne 2016. For more detail on this roadmap, watch Anil Gaur's section of the Java Keynote at JavaOne 2016. This survey was conducted to provide input and feedback on the component technologies included in the proposed roadmap.

The survey was opened on Friday Sept. 16, 2016. The survey was open to all users who wished to participate. The total number of surveys completed and submitted was 1693. The following graphics illustrate how survey responses were distributed across global geographic regions. Geographic location information is based solely on IP address of the computer used to complete the survey. As such, location data are only approximate.

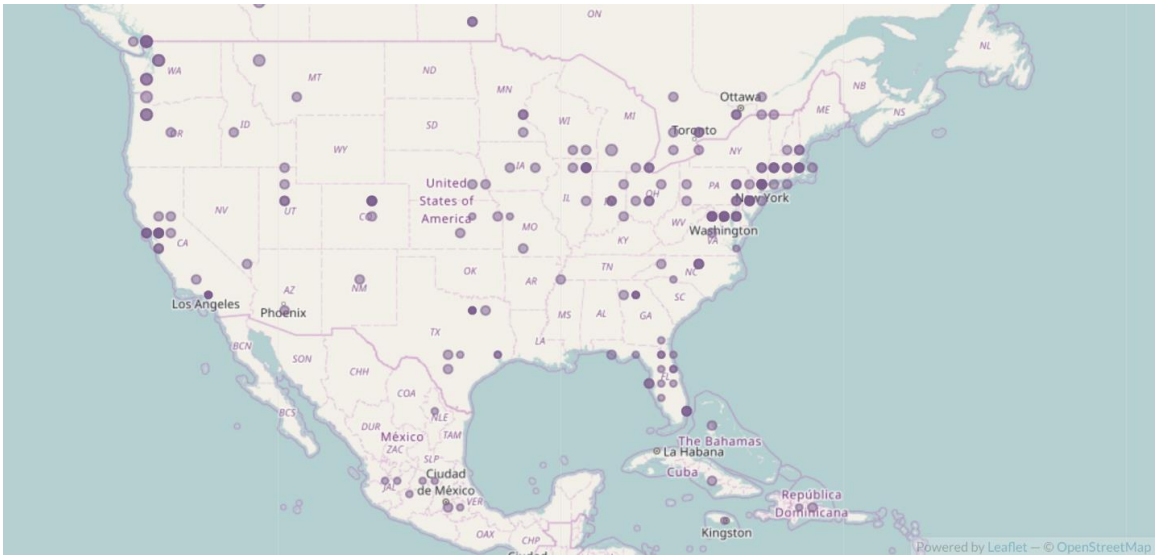
Survey worldwide geographic coverage:



Survey geographic response detail for European region:



Survey geographic response detail for North American region:



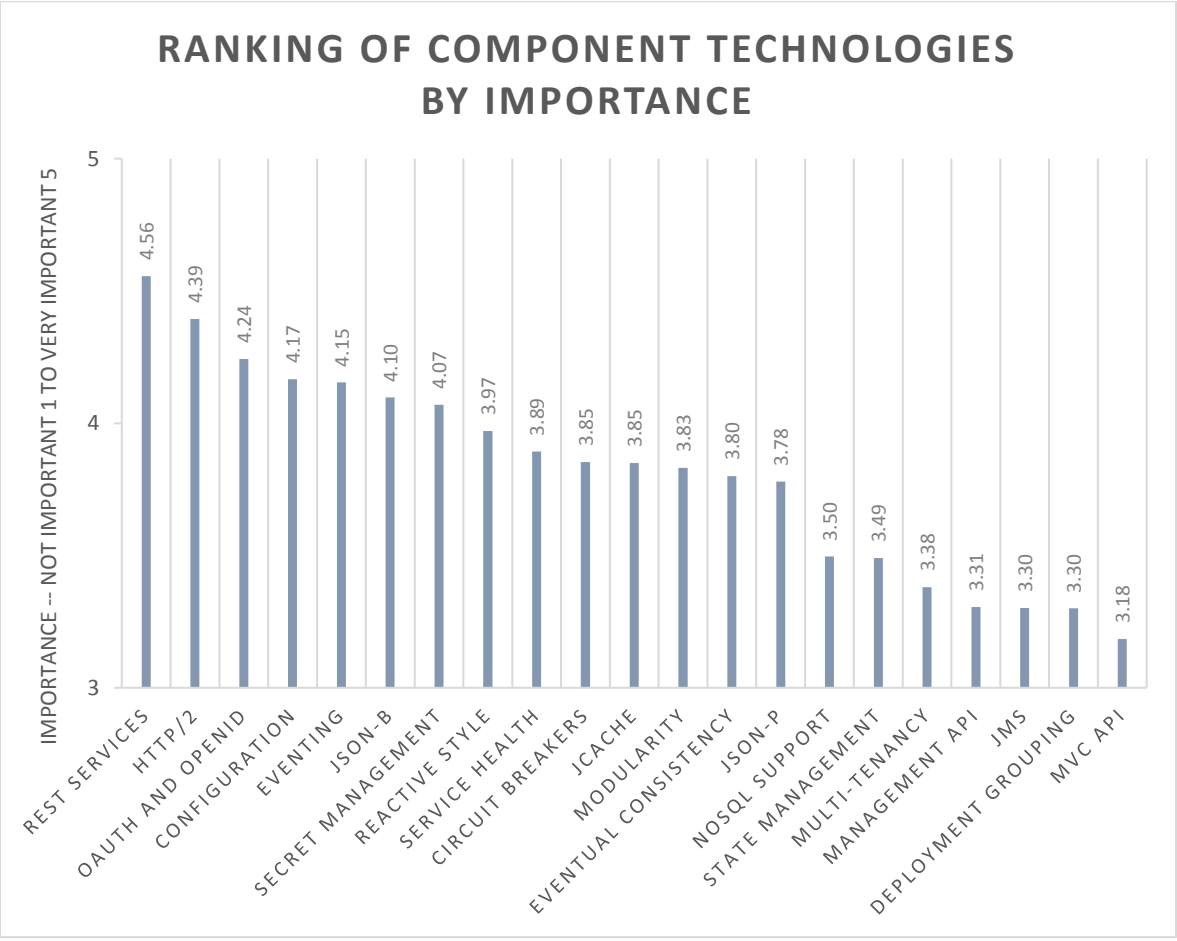
Ranking of Technologies by Importance

The survey consisted of 2 Demographic questions and 21 questions about proposed Java EE component technologies. The component technology questions asked respondents to rate the importance of the technology by selecting from the following: “1=Not, Important”; 2, 3; 4; “5=Very Important” or “Not sure.”

For each of the 21 component technologies, the average importance of the technology was computed by taking the average of all responses. If a response was “Not Sure” that response was not included in the average ranking. Detailed response data are provided at the end of this document, along with the text of each question.

Ranking Based on Importance from All Completed Respondents

The ranking of technologies, based on average importance, is given below:



Note that no technology averaged an unfavorable importance rating of below 3 in this survey.

Ranking of Technologies Based on Reported Experience

The ranking of technologies varies somewhat based on reported experience using Java EE or microservices technology, but not in a significant way that would alter our conclusions from the survey.

Rank order, by reported years of experience using Java EE is given below:

None (n=34)	0-2 Years (n=253)	2-8 Years (n=681)	More than 8 years (n=725)
REST Services	REST Services	REST Services	REST Services
HTTP/2	HTTP/2	HTTP/2	HTTP/2
Eventing	OAuth and OpenID	Configuration	OAuth and OpenID
OAuth and OpenID	Eventing	JSON-B	Eventing
Modularity	Configuration	Oauth and OpenID	Configuration
JSON-B	JSON-B	Eventing	Secret Management
Configuration	Secret Management	Secret Management	JSON-B
Secret Management	Reactive Style	Reactive Style	Reactive Style
Management API	Service Health	JCache	Service Health
Multi-tenancy	JCache	JSON-P	Circuit Breakers
Eventual Consistency	Modularity	Circuit Breakers	Modularity
JSON-P	Eventual Consistency	Eventual Consistency	JCache
Reactive Style	Circuit Breakers	Service Health	Eventual Consistency
Service Health	JSON-P	Modularity	JSON-P
State Management	NoSQL Support	NoSQL Support	State Management
JCache	State Management	State Management	NoSQL Support
NoSQL Support	Deployment Grouping	Multi-tenancy	Multi-tenancy
Deployment Grouping	Multi-tenancy	Management API	JMS
Circuit Breakers	MVC API	JMS	Deployment Grouping
MVC API	Management API	MVC API	Management API
JMS	JMS	Deployment Grouping	MVC API

Table 1 Rank order, based on reported years of experience using Java EE

Rank order, by reported years of Microservices Architecture Experience is given below:

None, No plans to start (n=350)	None, plan to start (n=495)	0-2 Years (n=620)	2-5 Years (n=194)	More than 5 years (n=34)
HTTP/2	REST Services	REST Services	REST Services	REST Services
REST Services	HTTP/2	HTTP/2	HTTP/2	HTTP/2
Application Configuration	OAuth and OpenID	OAuth and OpenID	Eventing	Eventing
OAuth and OpenID	Eventing	Application Configuration	OAuth and OpenID	OAuth and OpenID
JSON-B	JSON-B	Eventing	Reactive Style	Reactive Style
JCache	Application Configuration	Secret Management	JSON-B	Application Configuration
Secret Management	Secret Management	JSON-B	Application Configuration	Service Health
JSON-P	Reactive Style	Reactive Style	Secret Management	Secret Management
Eventing	Modularity	Service Health	Circuit Breakers	JSON-P
Modularity	Eventual Consistency	Circuit Breakers	Service Health	Eventual Consistency
Service Health	Circuit Breakers	Eventual Consistency	Modularity	JSON-B
Reactive Style	JCache	JCache	JCache	Modularity
Circuit Breakers	JSON-P	Modularity	Eventual Consistency	Circuit Breakers
State Management	Service Health	JSON-P	JSON-P	State Management
JMS	NoSQL Support	NoSQL Support	NoSQL Support	NoSQL Support
MVC API	State Management	State Management	State Management	JCache
Eventual Consistency	Deployment Grouping	Multi-tenancy	JMS	JMS
Multi-tenancy	Multi-tenancy	Management API	Deployment Grouping	Multi-tenancy
Deployment Grouping	Management API	JMS	Multi-tenancy	Management API
Management API	JMS	Deployment Grouping	Management API	Deployment Grouping
NoSQL Support	MVC API	MVC API	MVC API	MVC API

Table 2 Rank order, by reported years of Microservices Architecture Experience

Conclusions

We reviewed the Java EE 8 proposal based on these survey results, and additional review of implementation considerations. We have concluded that:

- REST (JAX-RS 2.1) and HTTP/2 (Servlet 4.0) have been voted as the two most important technologies surveyed, and together with JSON-B represent three of the top six technologies. Much of the new API work in these technologies for Java EE 8 is already complete. There is significant value in delivering Java EE 8 with these technologies, and the related JSON-P updates, as soon as possible.
- CDI 2.0, Bean Validation 2.0 and JSF 2.3 were not directly surveyed, but significant progress has been made on these technologies and they will be included in Java EE 8.
- We considered accelerating Java EE standards for OAuth and OpenID Connect based on survey feedback. This could not be accomplished in the Java EE 8 timeframe, but we'll continue to pursue Security 1.0 for Java EE 8.
- At JavaOne, we had proposed to add Configuration and Health Checking to Java EE 8, and these technologies rank reasonably high in survey results. However, after additional review we believe the scope of this work would delay overall Java EE 8 delivery. We have concluded it is best to defer inclusion of these technologies in Java EE in order to complete Java EE 8 as soon as possible.
- Management, JMS, and MVC ranked low in survey results, and this ranking supports our proposal to withdraw new APIs in these areas from Java EE 8. We have withdrawn the JSRs for Management 2.0 (JSR 373), and JMS 2.1 (JSR 368), and are investigating a possible transfer of MVC to another community member or organization in order to complete JSR 371 as a stand-alone component.

We will revise the Java EE 8 proposal consistent with these findings. The table below summarizes Oracle's original and revised Java EE 8 proposals, focusing on areas of new API development:

Original and Revised Java EE 8 Proposals

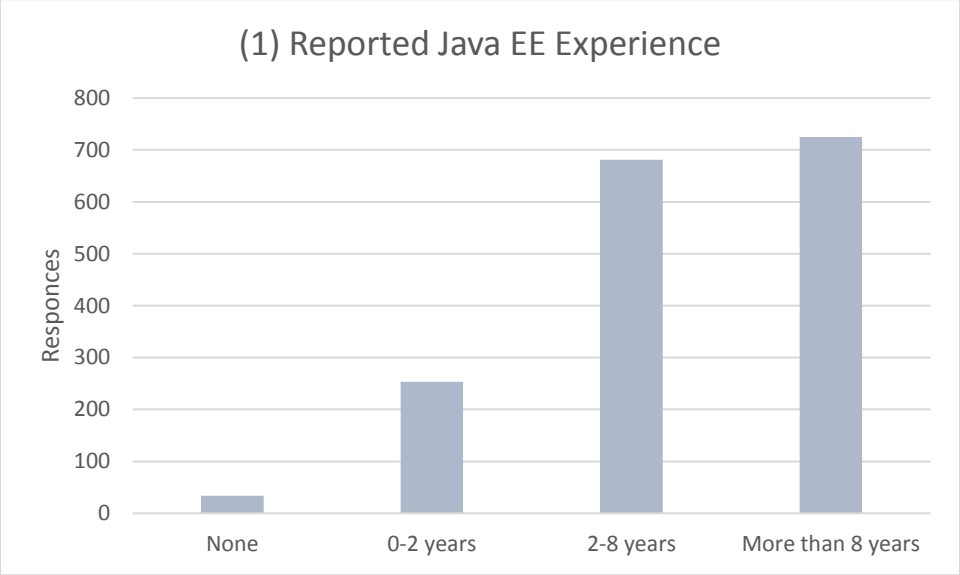
API from Original Java EE 8 Proposal	Survey Item	Survey Rank	Revised Java EE 8 Proposal
JAX-RS 2.1	REST Services	1	No change, retain
Servlet 4.0	HTTP/2	2	No change, retain
JSON-B 1.0	JSON-B	6	No change, retain
JSON-P 1.1	JSON-P	14	No change, retain
CDI 2.0	Not surveyed	N/A	No change, retain
Bean Validation 2.0	Not surveyed	N/A	No change, retain
JSF 2.3	Not surveyed	N/A	No change, retain
Security 1.0	Not surveyed	N/A	No change, retain
Management 2.0	Management	18	Withdraw Mgmt 2.0, Include Mgmt 1.0
JMS 2.1	JMS	19	Withdraw JMS 2.1, Include JMS 2.0
MVC 1.0	MVC	21	Withdraw

Detailed Survey Results

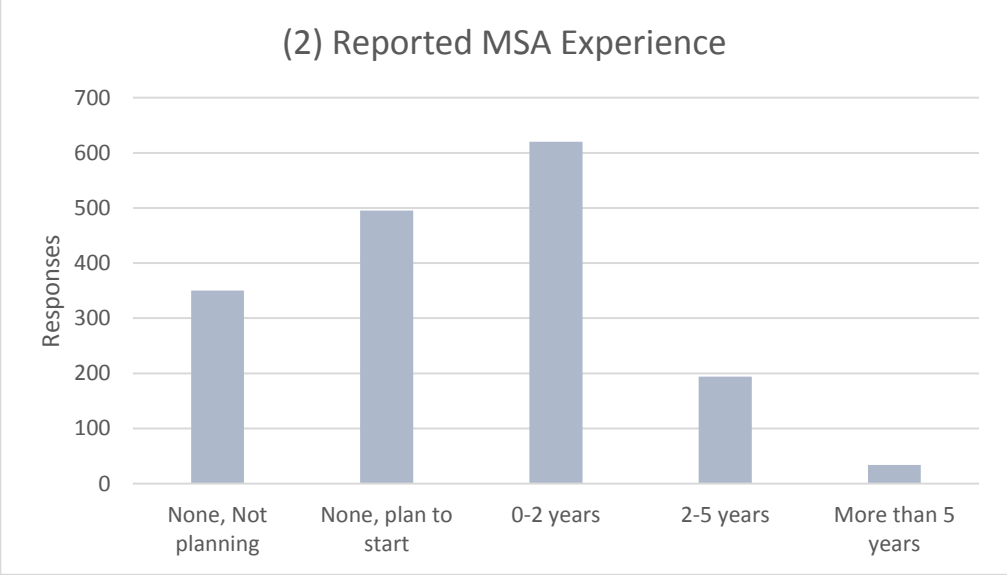
Survey Experience Questions

1. **How much experience do you have developing with Java EE?**

(None; 0-2 years; 2-8 years; more than 8 years)



2. **How much experience do you have developing microservices?**
   (None, no plans; None, but planning w/in next year or two; 0-2 years; 2-5 years; more than 5 years)



Technology Questions

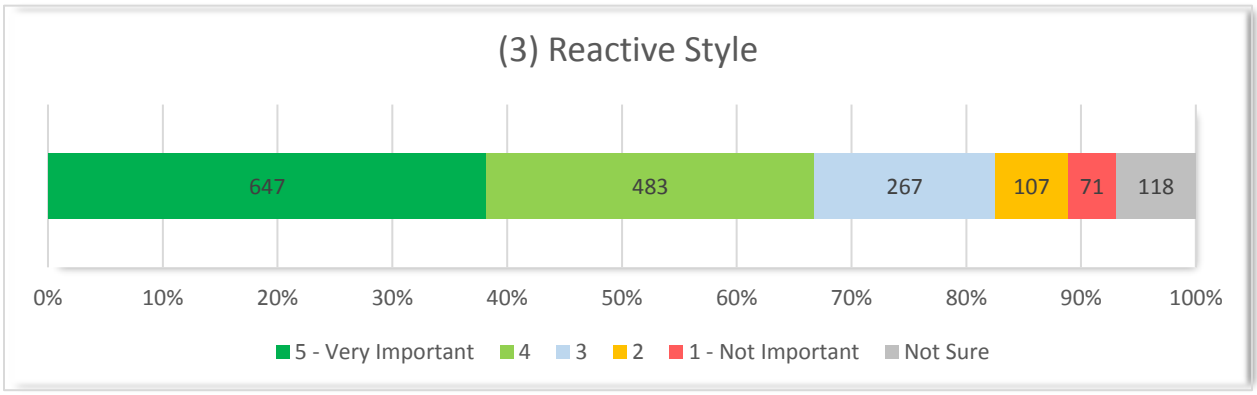
Programming Model
For many years, Java EE technology has been at the heart of enterprise application development. Recently, technologies associated with the cloud, such as containerization, microservices, REST, pay-per-use computing and continuous delivery have become more relevant. Enterprises have shifted from using application servers and their associated deployment artifacts to newer models that take advantage of the strengths of the cloud.

The following questions ask for your feedback on various changes under consideration for Java EE.

3. Reactive Style

While synchronous APIs are still used for the vast majority of applications running on JVMs, asynchronous, non-blocking programming models are gaining popularity, especially when developing in a microservice style architecture, where most, if not all, inter-service calls are remote by definition. In such scenarios, blocking calls can be very detrimental to overall performance, resource utilization, and scalability. Recent Java SE improvements (e.g. CompletableFuture , the new Flow API) enable a more Reactive approach.

How important is Reactive programming style support for the next generation of cloud and microservices applications?

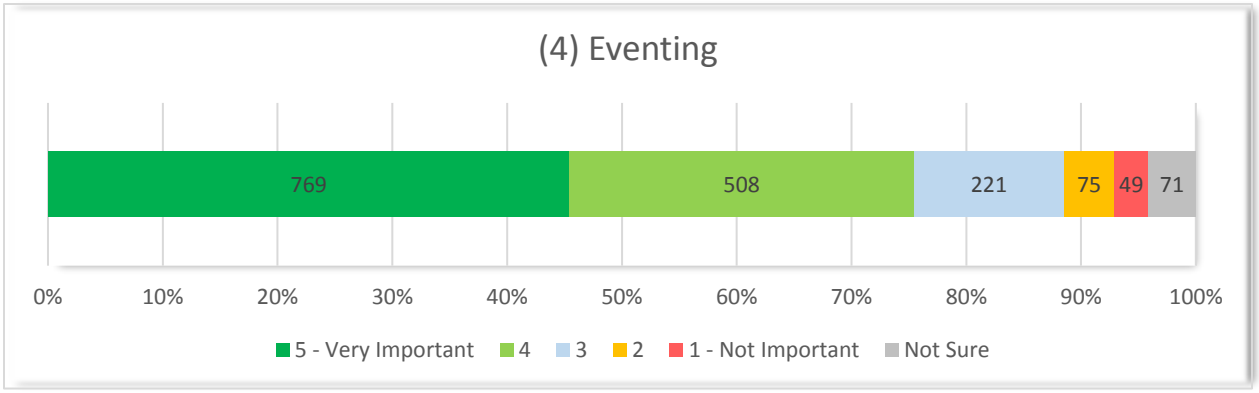


Rank across all responses: 8 / 21

4. Eventing

Many cloud applications are moving from a synchronous invocation model to an asynchronous event-driven model. Key Java EE APIs could support this model for interacting with cloud services. A common eventing system would simplify the implementation of such services.

How important is Eventing support for the next generation of cloud and microservices applications?

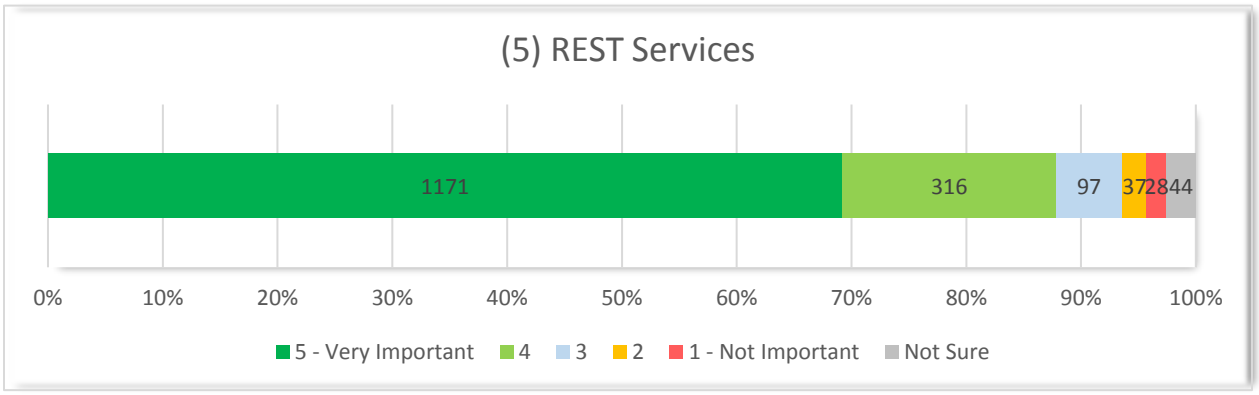


Rank across all responses: 5 / 21

5. REST Services

The current practice of cloud development in Java is largely based on REST and asynchrony. For Java developers, that means using the standard JAX-RS API. Suggested enhancements coming to the next version of JAX-RS include: a reactive client API, non-blocking I/O support, server-sent events and better CDI integration.

How important are the new features proposed for JAX-RS, for the next generation of cloud and microservices applications?



NOTE: due to an error, this question asked for importance between “0-Not important” to “4-Very Important”. The responses have been normalized for consistency with the other questions that rank from “1-Not Important” to “5-Very Important”

Rank across all responses: 1 / 21

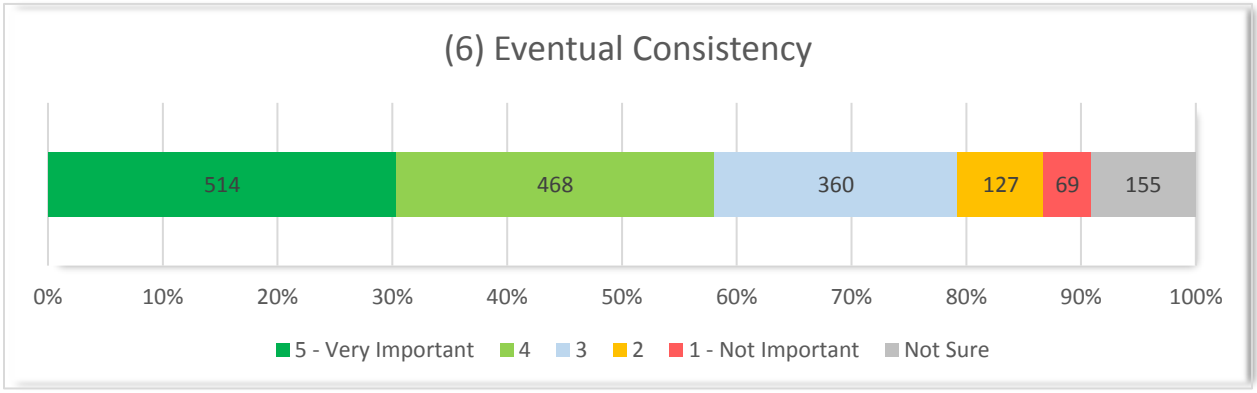
6. Eventual Consistency

Application development style is changing. Monolithic applications are evolving to have many smaller minimal function microservices that can be developed and managed independently. Each microservice is typically organized around business capabilities and may have its own datastore.

The programming model for eventual consistency across microservices is becoming an

important problem area. In this context eventual consistency may allow for microservices to observe other service's objects and state. Changes to the observed object would result in notifications to the observers. In some cases these notifications could automatically be batched. The proposal would aim to minimize the code both the observer and observable would need to develop in order to participate in eventual consistency for the microservice's targeted objects or state.

How important is eventual consistency support for the next generation of cloud and microservices applications?

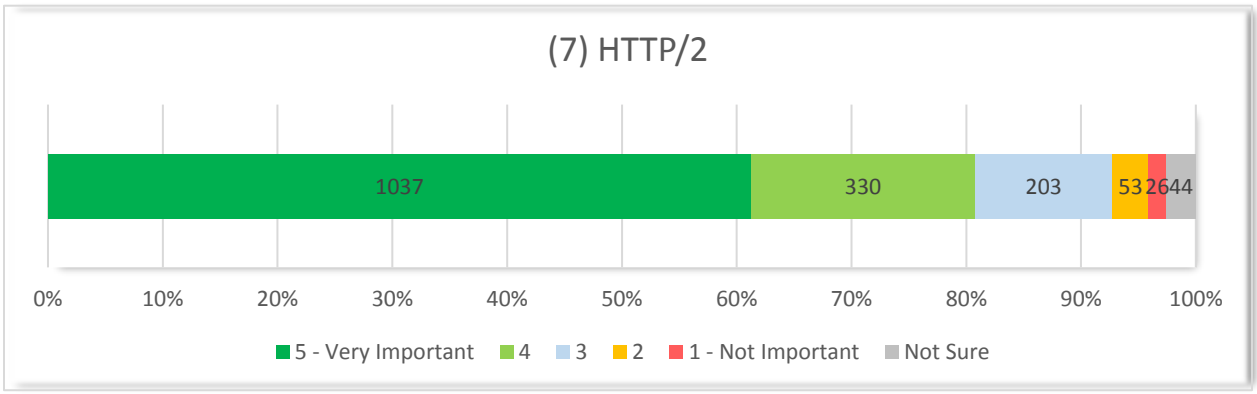


Rank across all responses: 13 / 21

7. HTTP/2

The HTTP/2 protocol enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients. On the client-side, HTTP/2 is now supported by all modern browsers and Java SE 9 plans to introduce a client-side HTTP/2 API. The Servlet 4 API plans to introduce server-side HTTP/2 support.

How important is HTTP/2 for the next generation of cloud and microservices applications?

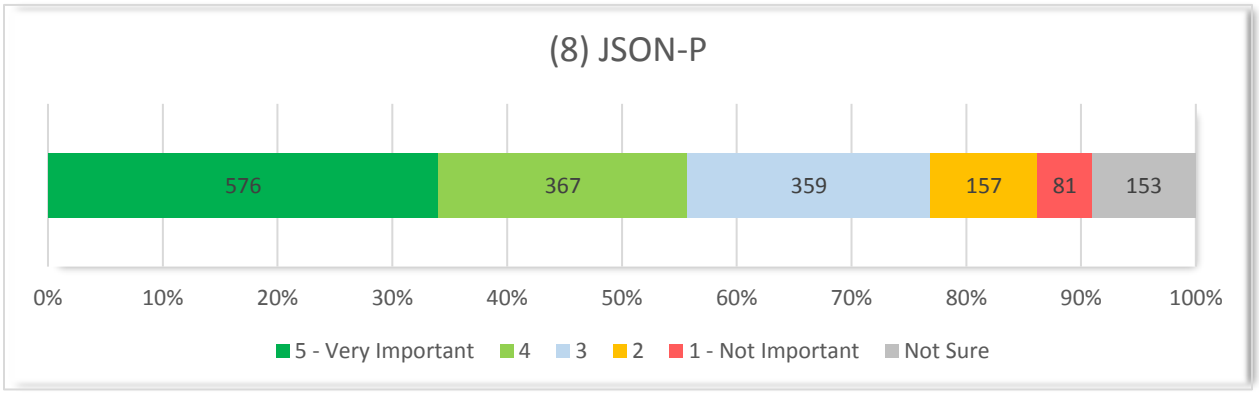


Rank across all responses: 2 / 21

8. JSON-P

New features in the JSON-P 1.1 API include support for new standards (JSON Pointer, JSON Patch and JSON Merge Patch), inclusion of JSON Collectors for the Stream API and other enhancements.

How important are the new features proposed in JSON-P for the next generation of cloud and microservices applications?

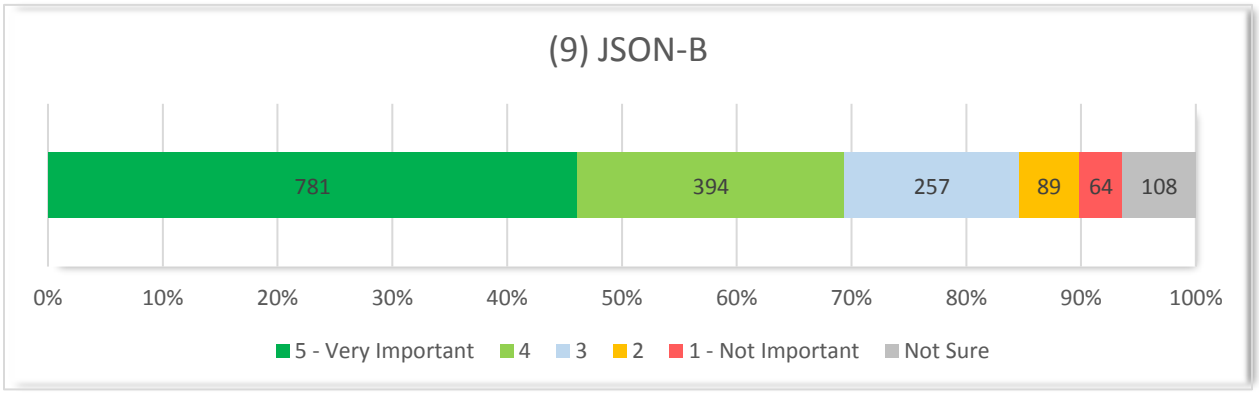


Rank across all responses: 14 / 21

9. JSON-B

To complete the support for JSON in the platform, Java EE needs an API to bind JSON documents to Java objects. This work is a new API (JSON-B), introduced in the proposed Java EE 8 release (JSR-367).

How important is JSON-B support for the next generation of cloud and microservices applications?

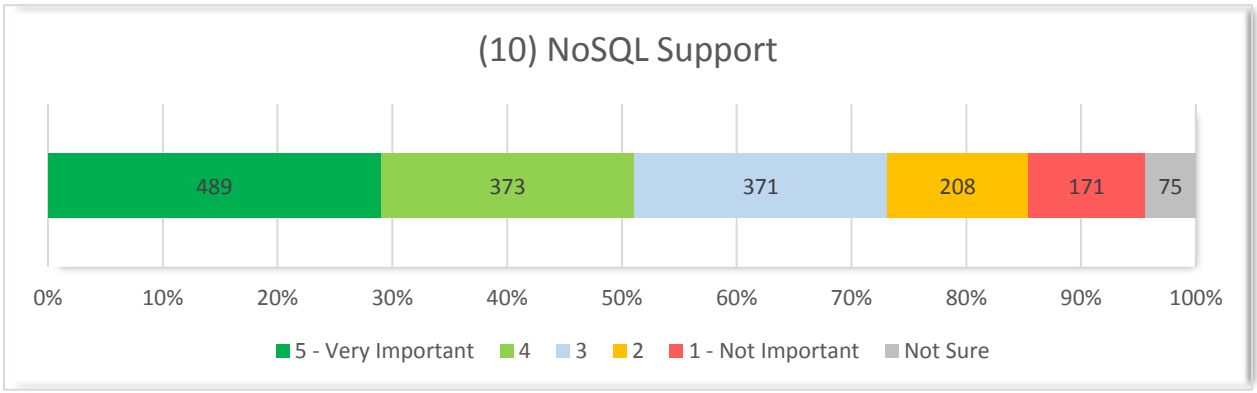


Rank across all responses: 6 / 21

NoSQL
10. NoSQL Support

Java EE has traditionally focused on standardization of APIs that access relational databases. Many applications are starting to choose NoSQL databases to store some or all of their persistent data. The databases may be used as replacements or additions to standard RDBMS storage. There are diverse categories of NoSQL providers. There is no standard APIs available for developers. There may be value in providing common abstractions for CRUD operations and additional support for the most common flavors of NoSQL database across categories (e.g. Key/Value, Document, Column, Graph). There also may be value in a simplified querying mechanism and an option for direct access to vendor specific APIs for applications that needs that unique functionality provided by specific vendors. Developers who stick to the standard APIs will be able to migrate their applications without any code changes to another vendor within the same category of NoSQL database (e.g. Key/Value to Key/Value).

How important is standardizing NoSQL database support for the next generation of cloud and microservices applications?



Rank across all responses: 15 / 21

Application Configuration
11. Configuration API

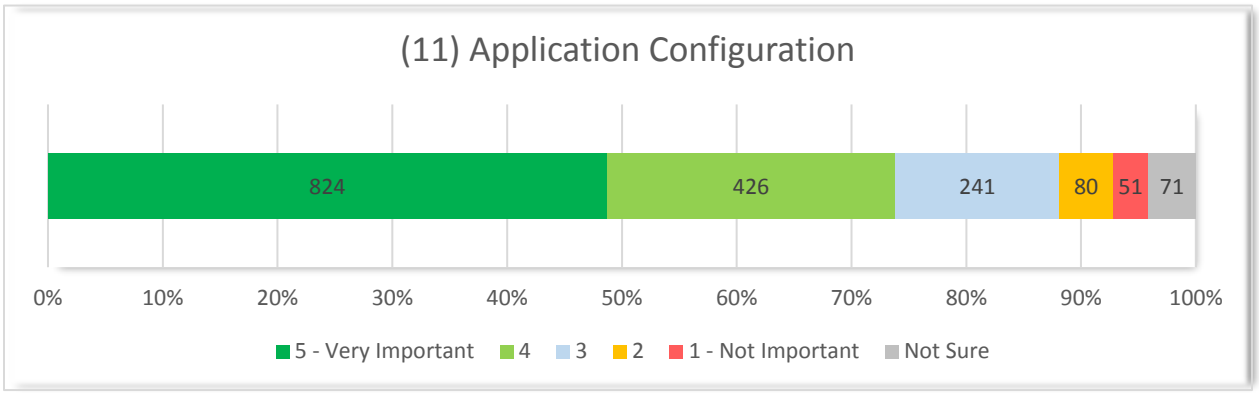
In a scenario where applications consist of services, some of which may be deployed to a cloud provider, Developers and DevOps engineers face many challenges related to managing application configuration:

- How to deploy an application in different environments without cracking its package?
- How to apply configuration for all deployed instances of an application without any redeployment?
- How an application can be notified if some of configuration properties are changed?

A configuration standard would define a unified configuration access API that would solve such problems. That standard would provide the ability to create one or more configurations that are independent of, and decoupled from the applications that use them. This standard may also include configuration file format, configuration layering, integration with different cloud providers

and other features that simplify applications' configuration management in the Cloud.

Should we standardize a Java EE application configuration API?



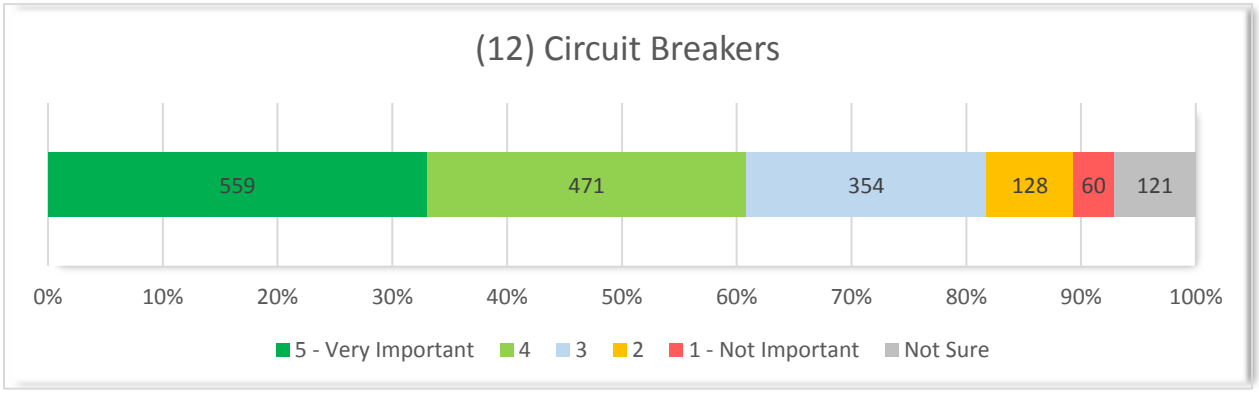
Rank across all responses: 4 / 21

Resilience

12. **Circuit Breaker**

In the Cloud, failure of application instances and services are inevitable. Applications need to be written to tolerate such failures, and not create cascading failures. A "circuit breaker" is a pattern which can be used to isolate and manage such failures. Key Java EE APIs could be updated to include support for circuit breakers, and in general provide better resiliency for network and service failures.

How important is "circuit breaker" support for the next generation of cloud and microservices applications?



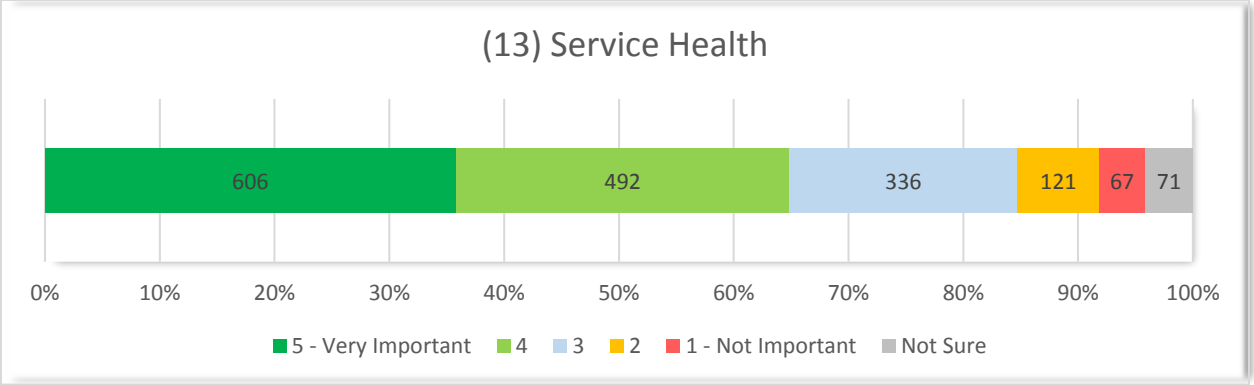
Rank across all responses: 10 / 21

13. Service Health

Cloud environments typically host large number of services; many of which are often inter-dependent. When problems surface, it is critical to quickly identify potential areas of failure to help fix the problem. Due to the scale, specific tools are often used to quickly narrow down the problem area.

Problems can be of various types, ranging from total failures, performance bottlenecks or other subtle issues which may be transient. Failures may also cascade to dependent services making the underlying cause hard to diagnose. Cloud platforms typically provide an Up/Down health check that only provides minimum information. In order for tools to provide an insight into underlying issues, a standards-based health check interface will be helpful. With such interface, service instances would publish their health information in a standard form so that a health monitoring system/service will be able to consume and analyze it uniformly.

Should Java EE introduce a mechanism to communicate the health of the cloud application to the cloud infrastructure?

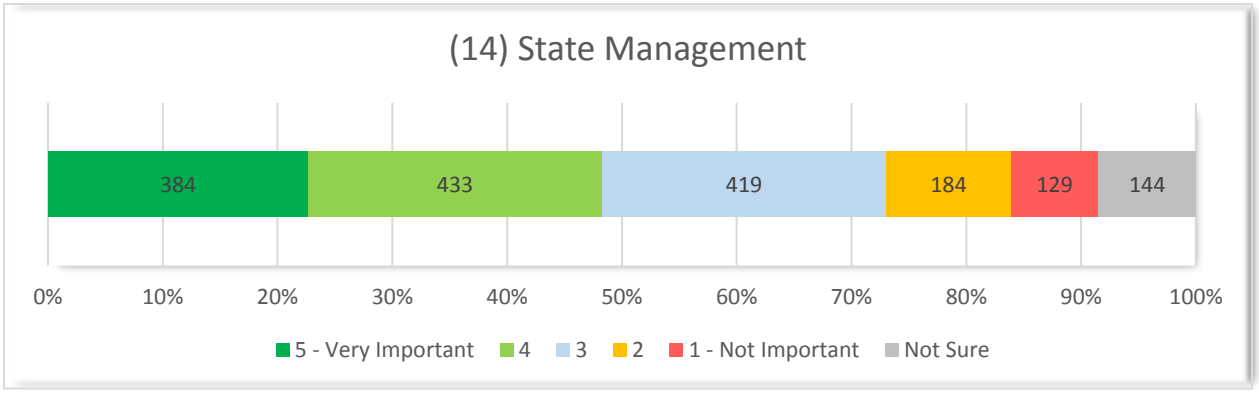


Rank across all responses: 9 / 21

14. State Management

Current trends talk about building 'stateless' applications and services, but the need to store some state exists nonetheless. This is obvious when building microservices, where each service must truly own its state. To be successful, many microservices need a scalable, fault tolerant state management solution.

Should Java EE investigate standards for state management?

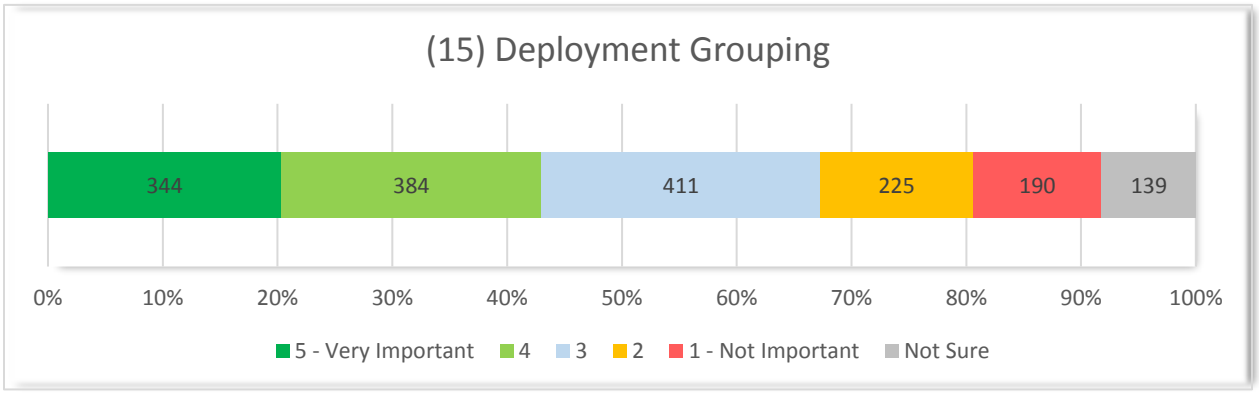


Rank across all responses: 16 / 21

Packaging
15. Grouping

Applications developed using a microservices architecture approach are composed of multiple services. While these services should be isolated, those same services may have some dependence relationships with other services to function properly. Java EE could define a packaging format that allows such a collection of services to be grouped together while specifying the dependencies and relationships between them, allowing convenient deployment and management of a group of services. This would be in addition to deployment and management of individual applications or services.

Should Java EE 9 investigate how to package a set of microservices together?



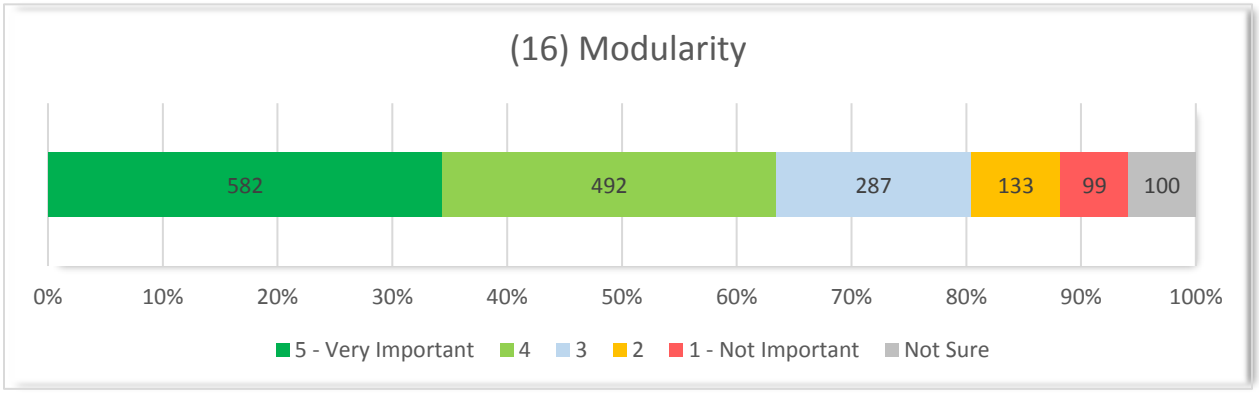
Rank across all responses: 20 / 21

16. **Modularity, Embeddability & Just Enough Runtime**

Many cloud applications are packaged and deployed as a stand-alone application executing in a Docker-like container. Java SE 9 modularity would allow us to deliver Java EE components as Java SE modules that could be used to create an application-specific runtime containing only

the modules needed by that single application. The Java EE runtime components could provide an "embedded" API allowing the user's application to have full control over the initialization and configuration of these components.

Should Java EE 9 investigate how to modularize EE Containers?

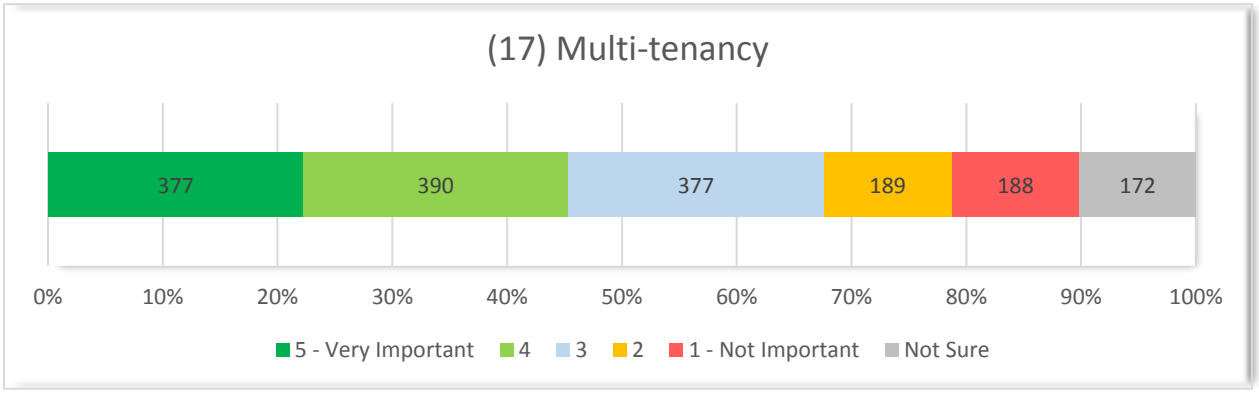


Rank across all responses: 12 / 21

17. **Multi-tenancy**

Cloud applications often serve the needs of multiple "tenants". Sometimes an application instance will be dedicated to a single tenant, and sometimes a single application instance will serve multiple tenants to better optimize resources. We could define how a Java EE container would support multiple tenants, and how an application would be configured for different tenants and would discover which tenant it is serving.

Should Java EE allow support for multi-tenant applications which could provide improved server density?



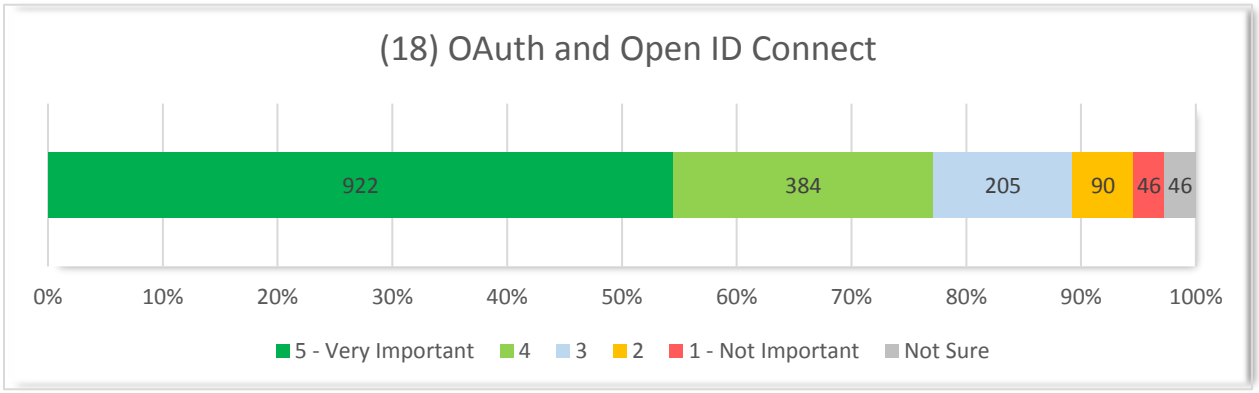
Rank across all responses: 17 / 21

Security

18. OAuth and Open ID Connect

OAuth and OpenID are seeing rapid adoption in cloud environments for authentication and authorization. We could enhance key Java EE APIs such as JAX-RS to better handle these technologies.

How important is OAuth and OpenID for the next generation of cloud and microservices applications?

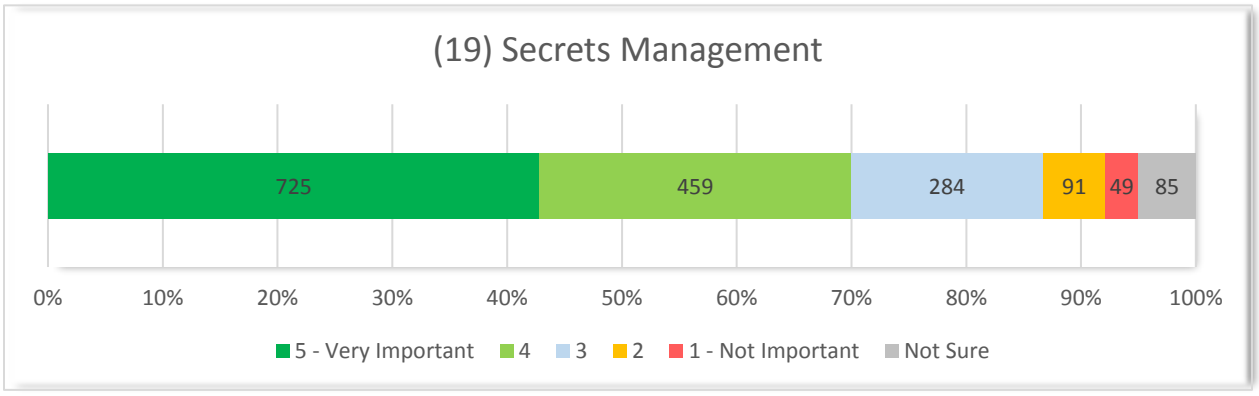


Rank across all responses: 3 / 21

19. Secrets Management

Cloud applications often need to access other services that require authentication, or require authentication of users of the application. Authentication is based on secrets, so applications need a way to store these secrets securely while allowing administrators to manage the secrets. We could define a secret management facility suitable for a cloud environment.

How important is secret management for the next generation of cloud and microservices applications?



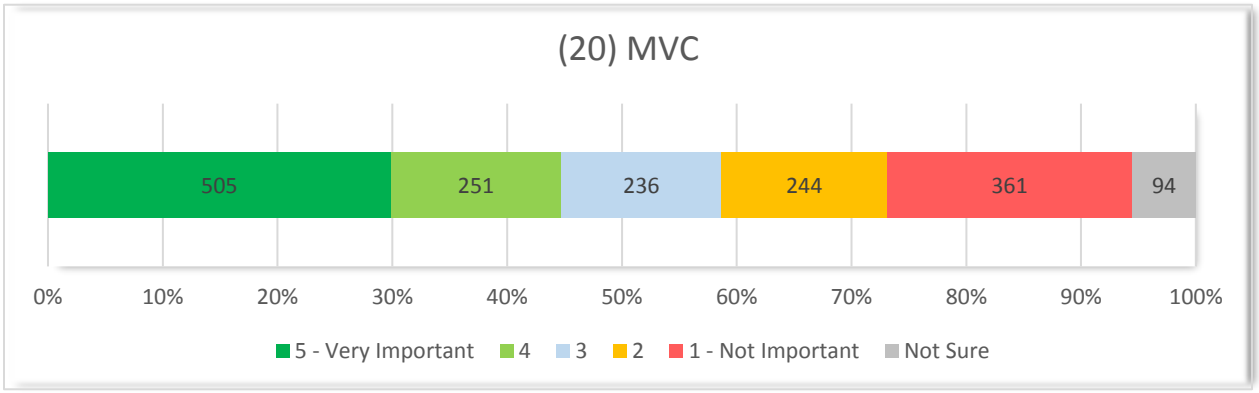
Rank across all responses: 7 / 21

Miscellaneous

20. MVC

When we first proposed Java EE 8, we got feedback that an action based web UI MVC framework standard would be a good addition to Java EE. At this point it seems that most new applications are using JavaScript based UI frameworks. We're now questioning whether it is still important to complete the MVC API (JSR 371).

How important is MVC API for the next generation of cloud and microservices applications?

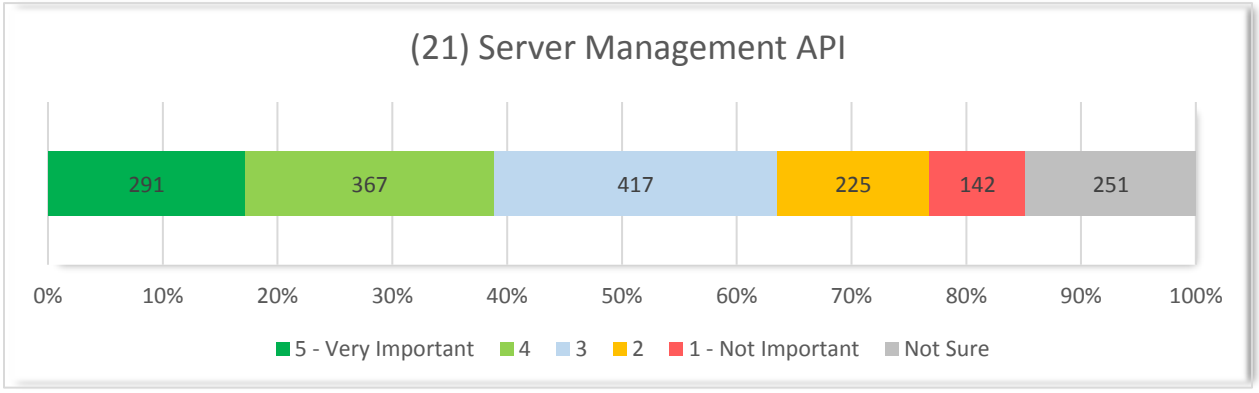


Rank across all responses: 21 / 21

21. Server Management API

Java EE includes an EJB-based management API. We proposed converting this API to a REST-based API in JSR 373, with little change in functionality. While a standard management API for Java EE applications in the cloud might be useful, as proposed, JSR 373 was not evolving to provide this functionality.

How important is the Management API, as proposed in JSR 373, for the next generation of cloud and microservices applications?

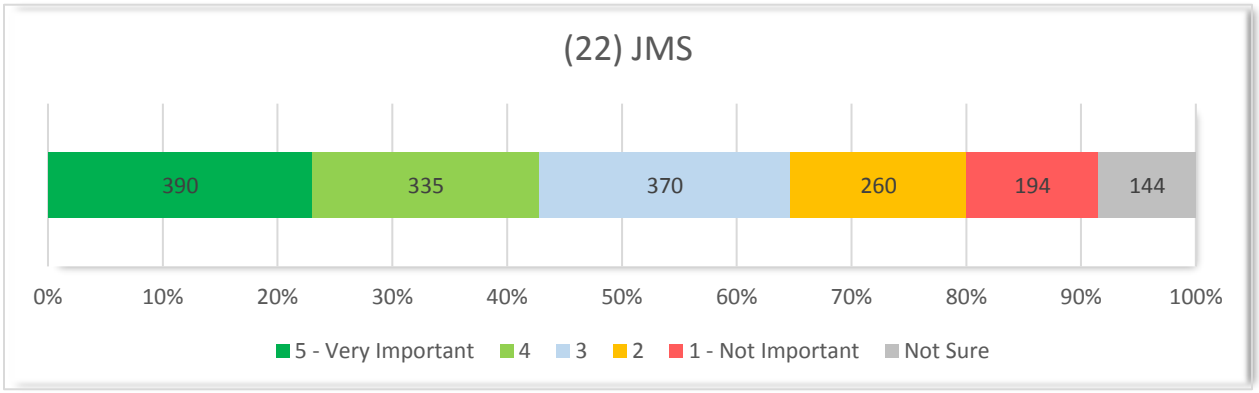


Rank across all responses: 18 / 21

22. JMS

JMS 2.0 introduced significant simplifications for Java EE 7. While further improvements are always possible, JMS is a relatively mature technology that does not appear frequently in cloud style applications.

How important is the continued evolution of the JMS API for next generation Java EE applications?

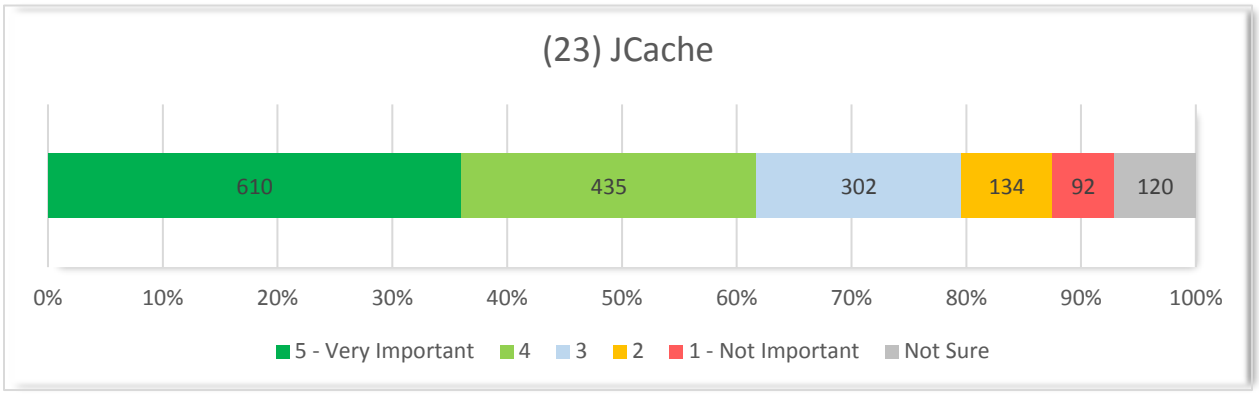


Rank across all responses: 19 / 21

23. JCache

JCache 1.0 provides often-requested pluggable caching mechanism for Java applications. We could integrate JCache with the Java EE platform.

How important is JCache for next generation Java EE applications?



Rank across all responses: 11 / 21