Java for the cloud, and the cloud for Java

Running Java in the cloud helps lower costs, speed up operations, and simplify the deployment.

by Leah Kulkhanjian

June 11, 2021

Many organizations are choosing to run existing Java applications in the cloud—and are increasingly developing new cloud-native applications in Java. Why is there synergy between Java and the cloud? Java Magazine recently talked with Dmitry Kornilov and Tomas Langer, two engineers who work on enterprise Java technology and have strong opinions on that topic. Kornilov, based in Prague, is director of software development at Oracle, and he oversees teams of Java developers working on Project Helidon and other enterprise initiatives. Langer, also in Prague, is a consulting member of Oracle technical staff, and he focuses on building out the Helidon architecture.

Java Magazine: Why use Java for the cloud?

Langer: One of the reasons why we use Java is because of the amount of support and services it provides. There are many libraries, technologies, and features that support developers, especially considering how long developers have been developing applications for the server side.

Secondly, since Java is a typed language, it has a huge advantage over untyped languages because of its maintenance. Java is less prone to runtime errors, and you can do static code analysis more easily. For example, if a company is looking for a language to run its applications on for the next 10 years, I would suggest Java, as it has the benefits I mentioned earlier, plus solid formatting standards and a whole ecosystem that allows companies to have an easy maintenance future.

Ultimately, the cost of the software is not in writing the code. The real cost of software is in the maintenance. Java has a huge
advantage in its IDEs, because it’s easy to add new features, fix bugs, and refactor environments if an organization needs to make changes.

_Java Magazine:_ How does Java, when you’re building for the cloud, compare to C, C++, C#, and newer languages such as Python?

_Langer:_ Writing code in C is very tough because of memory management. Even though there have been improvements in the language, you still have many language features that are really complicated. For example, if you need to do pointer arithmetic, you will need to take care of what is a reference, what is a value, and so on. Thus, the language is just much more complex. Also, if you make a mistake in C, the application will crash, the process ends, and your work is gone.

I think that C, C++, and similar languages are very good if you need to write something super small and performant and if you have people who actually understand those languages. However, now that there have been improvements in Java SE 9 and later versions, similar speeds can be achieved with the added advantage of being able to run the same binary on basically any system.

One huge difference from any other programming language is _Project Loom_ , which supports synchronous seemingly blocking code that doesn’t block threads. For example, if you write to a file system and it’s busy, your code will block, but the threads you were running on will be used by something else that doesn’t need to block. Once Project Loom is released, applications can get rid of things such as reactive programming (which would reduce scaffolding code) and the performance of applications will increase without many changes in the source code. No other programming language does this. From what we have seen so far from Project Loom, it will bring the synchronous world and nonblocking world seamlessly together, which will be a huge benefit.

Python has an advantage over Java when you’re writing simple applications because it’s quicker to write your first application or program. However, writing large applications on Python is a big formatting pain. Adding or removing one space in the wrong place will cause the whole program to collapse. This wouldn’t happen in Java because it’s fully functional even if you were to remove all spaces from the whole program. Java is much better for maintenance, troubleshooting, and discovering errors in code. I can take someone else’s Java code that I don’t understand and quickly reformat it in my IDE and read it.

_Java Magazine:_ When you’re building a new application, why deploy it in Oracle Cloud to start with?

_Langer:_ Nowadays if you want to write a performance application, you will use reactive programming. The current version of Helidon, _release 2.3.0_ , has a reactive integration with
Oracle Cloud Infrastructure. This means you have full reactive support on services such as object storage, vault, signature verification, encryption, and decryption. It’s always good to have a native implementation of that integration with the cloud, and this is something Helidon supports directly. Oracle Cloud Infrastructure has an SDK that is more suitable for blocking applications, but it’s something that is ready and more beneficial because the Java integration is maintained and complete.

Oracle Cloud Infrastructure provides a service for developers where they can point to a repository, which builds their application and releases it. For developers who are getting started with an application, this is very beneficial because even though there is less knowledge required to deploy to the cloud, they still need to understand some aspects of it if they were to go deep, such as using Kubernetes. So, the cloud is great for developers who are creating their first application or smaller scale applications.

*Java Magazine: Could you briefly describe Helidon and reactive programming?*

*Langer:* Helidon is a set of libraries designed for creating microservices-based applications. Its low-level set of libraries use reactive programming—meaning blocking operations are never used, leaving you to use the CPU as much as possible. That means you can have your application that is serving, say, 50,000 parallel requests using eight threads. Blocking a thread isn’t necessary while waiting for a database to return data. Similar to calling an external service, rather than blocking and waiting for a response, return the thread and when the bytes are available on the socket, continue the processing.

Although reactive programming is quite complicated and more error-prone when you’re writing blocking applications, it’s ideal for writing heavily parallel applications with super high performance. For example, it allows you to write trading applications that need to serve a huge number of parallel requests.

*Kornilov:* It’s about speed and performance. Helidon offers a mechanism to design and develop reactive services. Netflix and Apple services are both based on reactive services, although those companies don’t use Helidon. Nonreactive applications won’t see as many benefits. This indicates that bigger companies choose to use reactive services because of its advantages.

*Langer:* If a customer has a deployment of Java application servers in its environment and observes the CPU usage, it will discover that the most common production environments are running 10% to 20% of CPU utilization. To increase performance, usually the only choice is to add hardware and create a larger environment. However, with reactive programming, the CPU usage under load will be 100% as it’s not limited by blocking operations. So, if a customer wants to serve
millions of people at the same time, it will need a huge number of servers capable of doing that with a blocking API.

Helidon also has a runtime that is based on a blocking framework that uses CDI (Contexts and Dependency Injection) as the injection. It's called Helidon MicroProfile (Helidon MP), because it implements the Eclipse MicroProfile specification. This is the part where normal Java code is written with blocking operations. If you have a small number of parallel requests, there is absolutely no need to go reactive; you're able to do that with the regular blocking APIs, which are much easier to write and to read. So, that scenario is supported by Helidon MP.

With Project Loom, you will be able to write your application in Helidon MP in the blocking environment and have the benefits of reactive services. This is something I'm really looking forward to because it will make writing applications much easier. If the same performance is achieved with declarative blocking APIs, that will really help because it would remove 80% of all production issues with Java caused by deadlocks and an insufficient number of threads available due to too many threads running.

That's a summary of Helidon for writing microservices. Or, if you need to write macroservices, you can write big and complex applications, too. Helidon is like a single VM application that will serve your endpoints.

The Oracle Cloud Infrastructure integration is available for both the reactive and blocking worlds. Helidon also supports messaging and many other cloud services.

*Java Magazine*: Are there other reasons to run Java applications in the cloud?

**Langer**: Yes: services. Oracle Cloud Infrastructure has many services that require little coding by the developer.

Ultimately, companies don’t want to be an expert on anything outside their main business, and they shouldn’t have to. Companies want a technical service that can understand their business logic and transfer their business requirements into the service’s programming language. But everything else that can be a commodity should remain a commodity. Services on the cloud provide clear pricing, clear performance options, and straightforward results.

Another example is creating a vault for encrypted data. I don’t want to write, debug, or maintain security code, because as soon as I write security code, I’m opening myself up to security problems. If I need encryption, I don’t really want to have my own code encrypting things and then my own code decrypting things; I want to have a service for it. This is what Oracle Cloud Infrastructure Vault provides. I simply ask for the data to be encrypted by sending the data over the network to another service that will ask Oracle Cloud Infrastructure to decrypt it.
can trust that the algorithm will be strong and that I only need to configure my API towards the cloud.

Consider object storage: If large amounts of data need to be stored in a file that is one terabyte, this is a problem. Anybody who has tried it in history has failed, and developers end up having to rebuild the data file because they discover there are problems with temporary space, disk space, backups, and more. But when you are using a cloud service, storage becomes somebody else’s problem—meaning the problem disappears. The cloud is reliable and far more likely to find and fix all the problems encountered with tools and work as expected. This same capability to solve problems applies to any service from the list of cloud services available.

I always get people asking me, “How do you sort out load balancing?” I would say, “I don’t. That’s the cloud’s responsibility.” There are load balancers on the cloud that you can configure without needing to do any work. Oracle WebLogic has a full implementation of load balancing; it is actually in C++, because it’s part of the Apache plugin that provides that capability. That’s an example of how service providers simplify frameworks and implementations of applications and delegate them to cloud infrastructures.

When it comes to secured user accounts, there is a full implementation of LDAP (Lightweight Directory Access Protocol). All the security specs that you want to support in a project are there and don’t need to be manually implemented. The cloud provides a security service you’re able to redirect to, and it takes care of all the usernames, passwords, password recovery, two-factor authentication, and so on. So about 80% of the code you previously had to write is now all taken care of. This is why the cloud is the way forward. It’s like an application server that is moved from being something that you need to deploy yourself to something available with all the features ready.

I had an experience at my previous job where the cloud was helpful during testing. I was working for a company that was running an application in 10 countries, and each country had different requirements for the application. Before we moved to the cloud, testing a release required us to run 10 different full environments, which meant we needed to create the virtual machines, start them, and run them. We had hundreds of servers to take care of this, and we never knew what was in use and what wasn’t.

When application testing was moved to the cloud, the whole infrastructure worked in such a way that you would spin up your VMs, run the tests, and spin the VMs back down. This helped us realize how much the testing cost. Before, the utilization of the resources was bad because it was tough to keep track of all the servers and who was using them. The cloud made the accountability obvious but also the ease of use was very
different. A lot of the tooling and processes that usually needed to be manually designed were ready to use. This was a huge advantage for testing. Even though the deployment was performed in-house on a local data center, the testing was fully moved to the cloud.

Kornilov: That’s a good point. When we use the cloud, there are no limitations on how many resources we can use. In data centers, you are limited on the number of servers, whereas in the cloud, you will have access to more resources at the click of a button. This is great for on-demand workloads. As Langer explained, for testing, the cloud allows you to pay for testing servers and other resources during the months you are using the servers, and the rest of the year you can turn off the fee.

Dig deeper

- First steps with Oracle Cloud Infrastructure SDK for Java
- How to install Java in Oracle Cloud Infrastructure
- How to install and configure the Oracle Cloud Infrastructure SDK for Java
- SDK for Java documentation

Leah Kulkhanjian

Leah Kulkhanjian (@leakhulkhanjian), based near London, is a technology strategist for Oracle. She works with cloud engineers and customers to deliver cloud adoption frameworks and agile methodologies and analyzes cloud service consumption patterns.