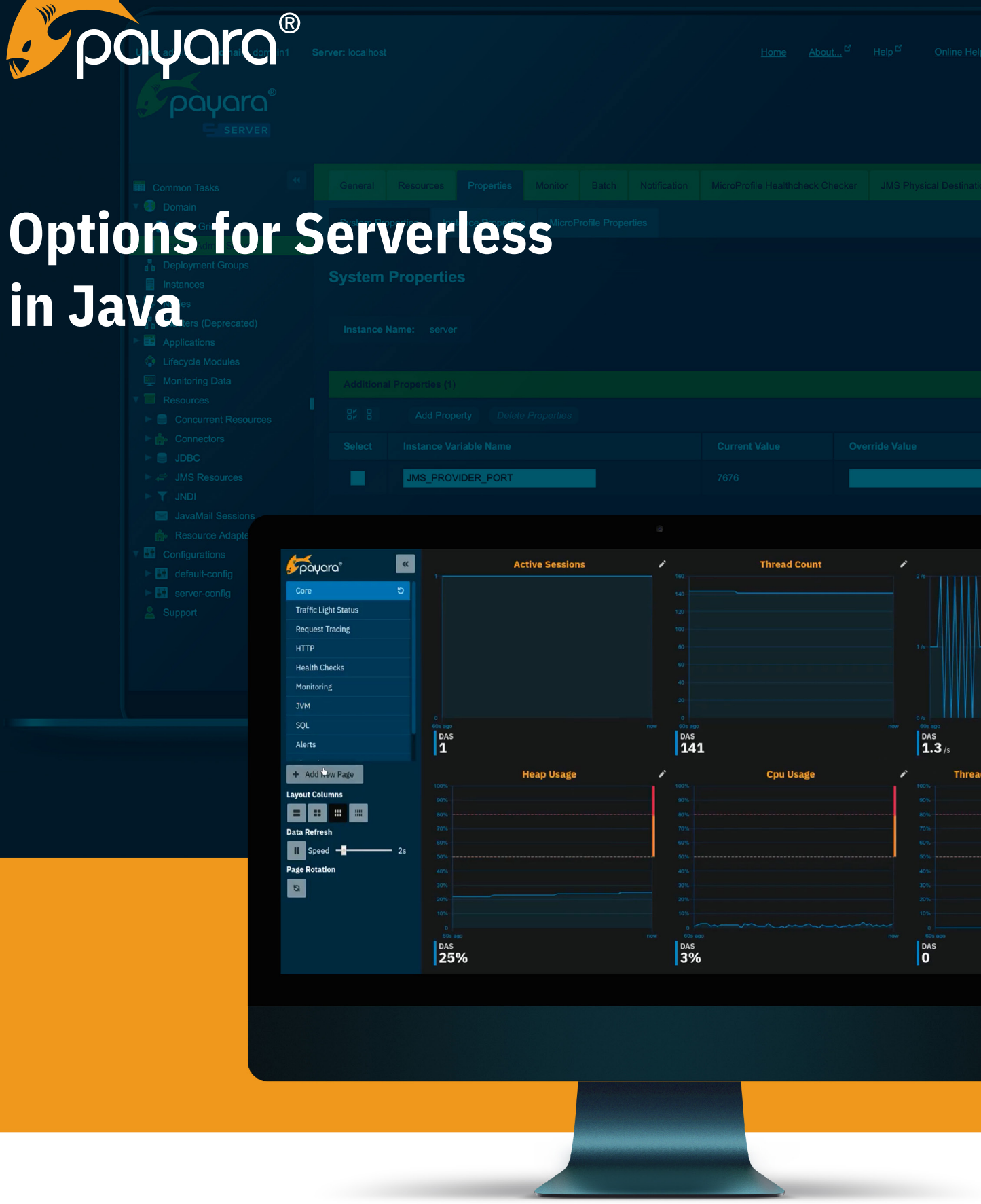
Options for Serverless in Java



The Payara® Platform - Production-Ready, Cloud Native and Aggressively Compatible.

eBook

Contents

Introduction

Serverless is a buzzword in software and the Java world. It is a cloud-native development model for building and running applications without managing servers. The cloud computing approach is growing in popularity in the Java space, and serverless is not far behind.

Related Java frameworks are working to meet this demand, like Spring Cloud and MicroProfile both developed for Spring Boot and Jakarta EE, respectively. These frameworks are designed to optimize Java development for microservices. Though a cloud approach is not necessarily reliant on a micro-services framework, the two often go hand in hand.

The 2022 Jakarta EE Developer survey revealed that the majority of the community plan to run over 80% of their systems in the cloud in the next two years. This means that a large proportion of the Java community are looking at migrating to the cloud and therefore may be evaluating serverless options to help them do so.

In this guide, we plan to explore serverless and provide information on different serverless options for Java, and the pros and cons of each approach.

Key Definitions

Serverless is an approach for cloud computing. You might want to use a serverless solution to save money and developer time when moving to a cloud native approach.

Serverless solutions for public cloud providers are events on demand through an execution-driven model. Therefore, there is no charge for unused serverless functions.

This in turn is likely to involve a microservices architecture, which will create specific demands related to the server functions needed.

First of all, then, let's straighten up the definitions that make up the serverless ecosystem.

Cloud Computing

The 'cloud computing delivery model' can be understood as infrastructure on demand; infrastructure that you don't manage, that can expand and contract, and that you can access, rapidly. This contrasts with the alternatives, such as an on-premise data centers or virtual machines, where rescaling takes longer and is more difficult to implement.

Cloud Native

Cloud native refers to a way of running applications that 'exploits the advantages of the cloud computing delivery model'. Cloud-native is, then, an approach that exploits this elastic, software-defined infrastructure style.

The Cloud Native Computing Foundation has a slightly differing definition, bringing in containers, service meshes and microservices. However, containers are an implementation of cloud native, and microservices are an application architecture model that does not have to use cloud.

Microservices

Microservices refers to a software architecture style where applications are structured by code in small, granular modules or services. A monolith application is split into a suite of small services. Services can then be deployed and maintained independently from each other.

Microservices don't have to run in a cloud environment necessarily. Microservices architecture can be built using on-premise data centres. And an application developed for the cloud can use a monolithic architecture!

However, a cloud environment is often easier for microservices, because it's easier to set up each new microservice/instance, and cloud environments tend to have specific management options for running several instances e.g. a centralised dashboard and automatic scaling.

It often makes sense for businesses to make the decision to move to a cloud-native environment at the same time as moving to a microservices architecture.

Server

A server is a wide, catch-all term that could be used to refer to a web server, an application server or an application runtime. You can find more detailed definitions of each in our guide, Beginners Overview Guide to Java Runtimes.

A server is technically a piece of software technology that simply provides a service to another computer program and its user. However, the way a server is understood in terms of web server, application server or runtime is as a piece of technology that handles infrastructural tasks needed to run an application.

When creating an application, you have the code related to the business logic; how data is created, stored and changed to model what the application actually does. Whatever your application does – anything from facilitating payments to allowing users to play a game, providing a software interface in a car to managing a smart home system – the business logic is responsible for make that happen.

You also need infrastructural code to manage all the other aspects of an application working; the background tasks that applications need to complete and may be less clearly related to the User Interface (UI) and what the application actually does. This includes integration with other systems, database connections, network related tasks and more. These are the tasks that are commonly dealt with by a server or runtime.

A web server handles HTTP requests. HyperText Transfer Protocol is a high-level protocol used on the Internet to communicate between different machines. Application Servers can handle HTTP requests but also a whole range of communication types, including Remote Method Invocation (RMI) and other types of API's and protocols like Java Message Service (JMS) and Simple Object Access Protocol (SOAP) for instance.

An Application Runtime indicates that you have a program that runs your application. It allows for further focus on the business logic only by handling tasks like logging, configuration, security and metrics.

From a Jakarta EE point of view, an application runtime or server also provides industry standard APIs and allow your applications to use them. Jakarta EE supplies the Java Persistence API, for example, which allows you to manage the persistence into a relational database by the use of underlying data source implementations within the application server. The Jakarta EE API also defines a standard for object/relational mapping and uses the data source to connect to the database.

A server can be used to refer to the software handling the tasks that a web server, application server or runtime might do. When it comes to moving to a cloud native approach, there are many more infrastructural tasks that must be managed. Add to the usual tasks, the provisioning of the Kubernetes resources, setting up the routing, networking aspects, providing the SSL certificate for your endpoints and more. This is where developers want to avoid the stresses of coding a server themselves – and may opt for serverless or allowing a vendor to take care of these tasks.

Serverless

Serverless is a term that is often misunderstood. It does not mean the lack of a server, but rather using a vendor that takes care of server issues on the behalf of their customers.

The issues related to running your applications – and to a cloud native approach – can be tackled by a vendor. Developers concentrate on developing the business-logic code and creating the application itself.

The extent to which infrastructural tasks are dealt with depends on the provider and technology of choice. We will now describe the different forms of serverless.

Function as a Service (FAAS)

FaaS is a type of cloud computing service that allows developers to build, run, and manage application packages as functions, without having to worry about maintaining the infrastructure. This approach requires your application to be composed of small components that respond to events. These components are called functions.

Functions contain only your business logic and the FAAS uploads them to the cloud. You then pay per function invocation. Here are some popular FAAS:

Amazon AWS Lambda

This is perhaps the most popular FAAS approach for Java users. It is provided by Amazon as part of Amazon Web Services. You can write and upload code as a .zip file or container image; an AWS Lambda will run it for you without having to worry about servers or clustering. It does so for multiple languages, including Java. Unlike Azure, Google and Oracle alternatives, AWS Lambda is considered 'edge-ready' - able to deploy closer to the data source.

Azure Functions

Azure Functions is Microsoft's offering for moving to a cloud computing model with FAAS. Again, you pay only for resources when your functions are running.

Google Cloud Functions

This is the FAAS offering if you are using Google Cloud. It is scalable and pay-as-you-go. You get credits to spend on functions as part of your cloud package.

Oracle Cloud Functions

You may be likely to choose FAAS because they integrate with the cloud environment chosen and other cloud services provided by Oracle, Google, Amazon or Azure. As the 2022 Stack Overflow developer survey reveals that Oracle Cloud Infrastructure is the least popular of the four, it follows that its FAAS is less popular. As with the others, you just write and deploy your code. Oracle will automatically provide and scale.

Advantages of FAAS

- **Cost-efficient**
  FAAS can work out incredibly cheap, as you don't need to over provision storage or compute resources for your application. You only must pay when the function is executed. This can vastly reduce operational expenses.
- **Resilience and failover are handled by the cloud**
  You may also save money in your teams, as these tasks, related to the system's ability to recover from a fault, are handled by the cloud.
- **No operating system administration knowledge required**
  Containers virtualize the operating system instead of hardware. With FAAS, you don't need to learn about administrating this virtual operating system.
- **No hardware skills needed**

Disadvantages of FAAS

- **Very little control over the execution of your application**
  You leave all the control over the execution of your application to the platform. Therefore, you lose some of the opportunity to introduce complex functionalities.
- **Limited in what APIs and libraries you have access to**
  You also need to be aware that this approach limits you to writing your application in specific languages and can only make use of certain APIs and frameworks. These are likely to be limited to those favored by your cloud provider of choice. They will try to lock you into a product suite. Check all your preferred technologies work with your chosen FAAS.
- **Event driven architecture needs different design patterns**
  FaaS architectures also tend to be event driven, which requires a very different architecture to the Object-Oriented approaches that are common today.
- **Difficult to test and debug applications**
  As all the infrastructure is managed by the cloud provider, it is harder to 'go in' to make changes.
- **No direct support for Java EE / Jakarta EE**
  Most cloud platform providers are not built with Jakarta EE – the specifications designed to help with Java enterprise applications specifically – in mind.

  Building your application for FaaS often severely limits how you can use Jakarta EE and a runtime. It's possible to run embedded microservices-oriented runtimes from a function but it's not straightforward and it's beneficial only for very complex functions.
- **You have to significantly adjust your Jakarta EE application for effective use with FAAS**
  The best usage of Jakarta EE applications with FAAS is to use FAAS functions only for specific application components which benefit from FAAS mostly, while building other components as Docker containers and running them on an automatic provisioning service. This means rewrites and adjustments.

- **You will still have to undertake significant work to prepare your application**
  With AWS Lambda, you still have to build an enterprise environment, a VPC (Virtual Private Cloud), a JAX-RS endpoint, and an API gateway, at the very least. You may be better suited to a solution like Payara Cloud, that takes advantage of the way Jakarta EE builds a ready-to-deploy application, separated from infrastructure.

Automated Cloud Provisioning Service

This is a cloud service that automatically provisions the necessary runtime. With these services, you upload your application to the cloud, and the service provisions appropriate infrastructure in a way that provides load balancing, scaling, and failover for you. This, in terms of automating best practices, ensures that your application meets your non-functional requirements.

Amazon Elastic Beanstalk

Amazon Elastic Beanstalk (EB) aims to simplify deploying applications to the cloud as much as possible. For Java applications, it provides a general Java platform to run any Java application, and a Tomcat platform to run applications based on plain servlets. Besides Java-specific platforms, it also supports platforms that can run any application packaged as a Docker container.

It allows creating a simple Docker environment and running any Docker container that is either built using a Dockerfile or downloaded from the central Docker Hub or from a private AWS repository hosted in the AWS Elastic Container Registry (ECR).

Cloud Foundry

Cloud Foundry is open source and based on Kubernetes. Developed by VMware, it is now owned by the Cloud Foundry Foundation. You deploy using the cf-push command and it uses four key Kubernetes technologies – KubeCF, Eirini, Quarks and BOSH – to get applications running on a Kubernetes cluster.

Heroku

Heroku manages your apps inside its 'dynos' - smart containers. It is owned by Salesforce and uses AWS technology.

Google App Engine

Google App Engine allows you to build and run your applications on Google's servers and offers automatic scalability. The important thing about Google App Engine is you do not have to containerize your apps – though support for this is included.

Advantages of Cloud Provisioning Services

- **You don't need as detailed architectural skills**

  You don't have to retrain your existing developers in the skills needed to provision the runtime. The automated configuration provided by services like Amazon Elastic Beanstalk means you might also reduce the chance of small mistakes when developers do it by themselves.
- **Provision of the infrastructure you need**

  You only have to pay for what you use, making it a cost-effective system. You match your server needs to your service load, and auto-scaling functionality means that extra servers are only brought up when needed.
- **Don't need as many systems administration skills**

  As above, you don't need to retrain in the administration skills needed to manage the application when it is already on the cloud.
- **Don't need detailed hardware skills**

  There is no hardware involved in this solution, so both the skills and the operational costs involved in hardware maintenance and security are eliminated.
- **Resilience and failover are automatically provided**

  As with FAAS, your application's ability to cope with failures is also dealt with by the solution.

Disadvantages of Cloud Provisioning Services

- **Often limits framework choices or packaging format (Docker)**

Cloud provisioning MIGHT limit you to working with a certain set of frameworks and containers. However, most of these services support running applications packaged as Docker containers, which allows using any frameworks and dependencies of your choice. Something like Heroku can be criticized for being too constraining – it may not be usable with older integrated technologies that companies may not be ready to move on from yet.

- **Still some resources provisioned even if not needed**
- **No direct support for Java EE / Jakarta EE or Payara Platform**

Even though you can use Jakarta EE runtimes through Docker, this still leaves the extra step of setting up Docker images. A solution like Payara Cloud would further minimize your infrastructural tasks by taking this out of your hands as well.

Managed Container Services

With the rise of the Container deployment model, a model has emerged to support the users who want to run their containers. It is very similar to the cloud provisioning services solution but in this case, you need to provide your container image that needs to be executed, just as you provide the application in the provisioning model. The provider is responsible for having an environment for you ready where they can run your image successfully. This is sometimes called Container as a Service (CAAS). Container engines, orchestration and the underlying computer resources are delivered to users as a service from a cloud provider.

AWS Elastic Container Service (ECS)

ECS allows creating any custom infrastructure based on Docker containers and provides orchestration of the containers, scheduling them on underlying virtual machines, scaling them based on defined rules, defining network topologies and everything you would need from a complex network infrastructure.

AWS Elastic Kubernetes Service (EKS)

EKS provides similar functionality based on the widely used Kubernetes project and thus allows a more standardized way of deploying and managing Docker-based infrastructure. EKS is recommended for more complex and standardized clusters of Docker containers. If you don't need features specific to ECS, it's a more convenient way to run clusters than ECS because it's more widely used, covered with a lot of documentation and guides and with more flexible configuration and services.

AWS Fargate

Fargate is a service that allows you to run Docker containers directly, without creating clusters or managing EC2 instances to run the containers. It's not very convenient to use it directly because most applications would need to run on multiple Docker containers or would at least benefit from scaling a single Docker image to multiple instances behind a load balancer. But this service is very powerful when combined with other AWS services that provide an abstraction over Fargate. For example, ECS cluster can use Fargate to run individual containers. Beanstalk Multi-container environment uses an ECS cluster, which can again use Fargate.

You still need to create your Docker containers through ECS and EKS, therefore we have included it under the managed container system rather than cloud provisioning services – which do have the potential to run without Docker, if not for Jakarta EE.

Google Cloud Run

Google Cloud Run abstracts away infrastructure management, but unlike Google App Engine, assumes the use of containers. You don't have to create a cluster or manage infrastructure, but your app will be containerized. Google Cloud build packs are available and can automatically build container images from source code. However, this will not be possible for Jakarta EE applications – for Jakarta EE, you still have to create a Kubernetes cluster.

Advantages of Managed Container Services

- **Straightforward for people who are already comfortable with Kubernetes**

  If you've already trained in Kubernetes, the process of adopting managed container services will have lower barriers to entry – there also shouldn't be a steep learning curve when moving between different managed Kubernetes vendors.

- **Can add more complexity than some FAAS or cloud provisioning services**

  For example, with AKS and ECS you can specify a particular network topology of Docker containers. You can create clusters of containers, with services, tasks and network configurations.

Disadvantages of Managed Container Services

- **More complex and time consuming than FAAS or cloud provisioning services**

  ECS or EKS for example, are much more difficult to set up than Amazon Elastic Beanstalk and a lot of functionality provided by Beanstalk out of the box needs to be enabled and customized.

- **In most cases, you have to create Docker images**

  Your engineers will still need to be trained in Docker and Kubernetes, even though infrastructural tasks will be dealt with. This is where a solution like Payara Cloud is useful – this will not only deploy the application on Kubernetes but complete all YAML configuration, actually build container images and create a pod.

- **No direct support for Java EE / Jakarta EE or Payara Platform**

  As mentioned above, even when Managed Container Services proclaim to be able to create container images for you, this will not be possible whilst also taking advantage of Jakarta EE. You can use them, but you will have to create your own images.

Jakarta EE and Serverless

Jakarta EE works differently to some of its competitors, such as Spring. Jakarta EE is packaged into an Archive file and this application is stand-alone. Your EAR or WAR contains application code, third party application dependencies and, optionally, deployment descriptors.

Therefore, a key Jakarta EE concept is a deployable application, isolated from infrastructure. It makes sense to extend this for a cloud native world – also automating the processes involved with moving to an external, cloud native infrastructure.

This is why Jakarta EE, a set of industry-standard APIs, prides itself on being ready for cloud-native Java. This includes the development of MicroProfile, a set of specifications that build on the Jakarta EE specifications and are designed to optimize Enterprise Java for a microservices environment.

Jakarta EE vendors are able to innovate and create options that take advantage of the way Jakarta EE builds a ready-to-deploy application, separate from infrastructure.

Payara Cloud

Payara Cloud is a serverless option, but this time the majority of infrastructural tasks are managed for you. All you need to do is deploy your WAR file, as you would deploy to your Jakarta EE runtime - only here, the tasks related to a cloud native approach are also dealt with.

Payara Cloud runs on Microsoft Azure, building a runtime on Azure Kubernetes Service. The software takes the user's WAR file, packages it into a Docker image with Payara Micro – our microservices

runtime – and completes all YAML, builds container images, creates a pod, deploys it on Kubernetes, updates the API server to manage ingress on Microsoft Azure and even creates an SSL certificate for the application.

You can then integrate this with your external Kubernetes platform – Kubernetes itself, OpenShift, or Rancher for example. Serverless has become more comprehensive, with more of the infrastructural tasks handled.

Advantages of Payara Cloud

- **No need for your development team to train in Kubernetes, Docker and other cloud native skills**
  Payara Cloud builds on the 'write once, deploy anywhere' philosophy of Jakarta EE. Payara Cloud scans your application for database usage and configuration parameters defined using the MicroProfile Config specification, and then presents you with a configuration screen to enter these values. That's all you need to do to connect to your database and deploy the application – shielding your developers from infrastructural tasks and meaning you don't have to spend time and money on learning Kubernetes and Docker.
- **Direct support for Jakarta EE**
  Unlike other serverless solutions, Payara Cloud uses Jakarta EE specifications. This means you don't have to worry about support for your Jakarta EE applications or to containerize your Jakarta EE applications in order to use the serverless technology – it is literally designed for you! This makes it more simple to use than other serverless options, if you know Jakarta EE; Java EE / Jakarta EE expert Adam Bien said "If you know a little bit about Java EE, Payara is a lot simpler than AWS Lambda."

- **Integrate with your CI/CD workflows**
  Payara Cloud provides a friendly user interface which allows your application to run in a managed cloud environment. While this is very convenient for configuration and troubleshooting work, integration in continuous deployment pipelines calls for something else. Payara Cloud is ready for this, with the option to deploy your application using a GitHub Action Workflow and Payara Cloud Command Line (PCL).

- **Faster and easier to use than other serverless options**
  Because Payara Cloud truly takes away almost all infrastructural tasks, it makes moving to the cloud as simple as pressing a button! Again, Adam Bien said, if you know Jakarta EE, "I would say, creating Lambda from scratch and shipping it would take you at least half an

hour, if not two hours. But to ship a Payara Cloud application, I would say 5 minutes. Because you only need a WAR."

- **Don't need detailed hardware skills**
  There is no hardware involved in this solution, so both the skills and the operational costs involved in hardware maintenance and security are eliminated.
- **Strong monitoring and diagnostics**
  You can view metrics of all applications within the namespace in one place, get detailed metrics of an individual application and diagnostic information will be provided for failed application deployments.

Disadvantages of Payara Cloud

- **Your application has to follow the Jakarta Web Profile specification**
  If you are not already using the Jakarta Web Profile specification, you will have to rewrite it. However, Jakarta EE is a widely adopted set of specifications extending Java SE - the standard edition Java programming language - with ways to perform the functions particularly useful for an enterprise application. With the recent release of Jakarta EE 10 under the Eclipse Foundation, the project is alive and creating new innovations by the second – it might be time to consider a move!
- **New technology – features still to come!**
  For example, clustering capabilities are currently in development for a future release of Payara Cloud. Similarly, Payara Cloud is currently a cloud-only product by design with a possible on-premise version to be developed in the future, and rolling upgrades are currently in development to eliminate downtime of your application during upgrade.

Summary

We hope that this guide has been helpful for both helping to understand what serverless is, and the many options that are available to you as a Java EE / Jakarta EE developer.

We hope to have shown that serverless isn't a single static concept with different competitors offering the same solution, under different brands. Instead, serverless is a catch-all term that can cover many different technologies, and there are pros and cons to how much you decide to offload your

infrastructural tasks. Often, you have to sacrifice elements of control in order for tasks to be taken out of your hands.

For ultimate simplicity and Jakarta EE support, Payara Cloud is an exciting new option that is being championed by Java EE/Jakarta EE experts. Find out more here.

Further reading:

- Payara Cloud Documentation
- Ignore Infrastructure and Concentrate on Code with Jakarta EE and Payara Cloud
- Payara Cloud Datasheet
- Explaining Microservices: No Nonsense Guide for Decision Makers