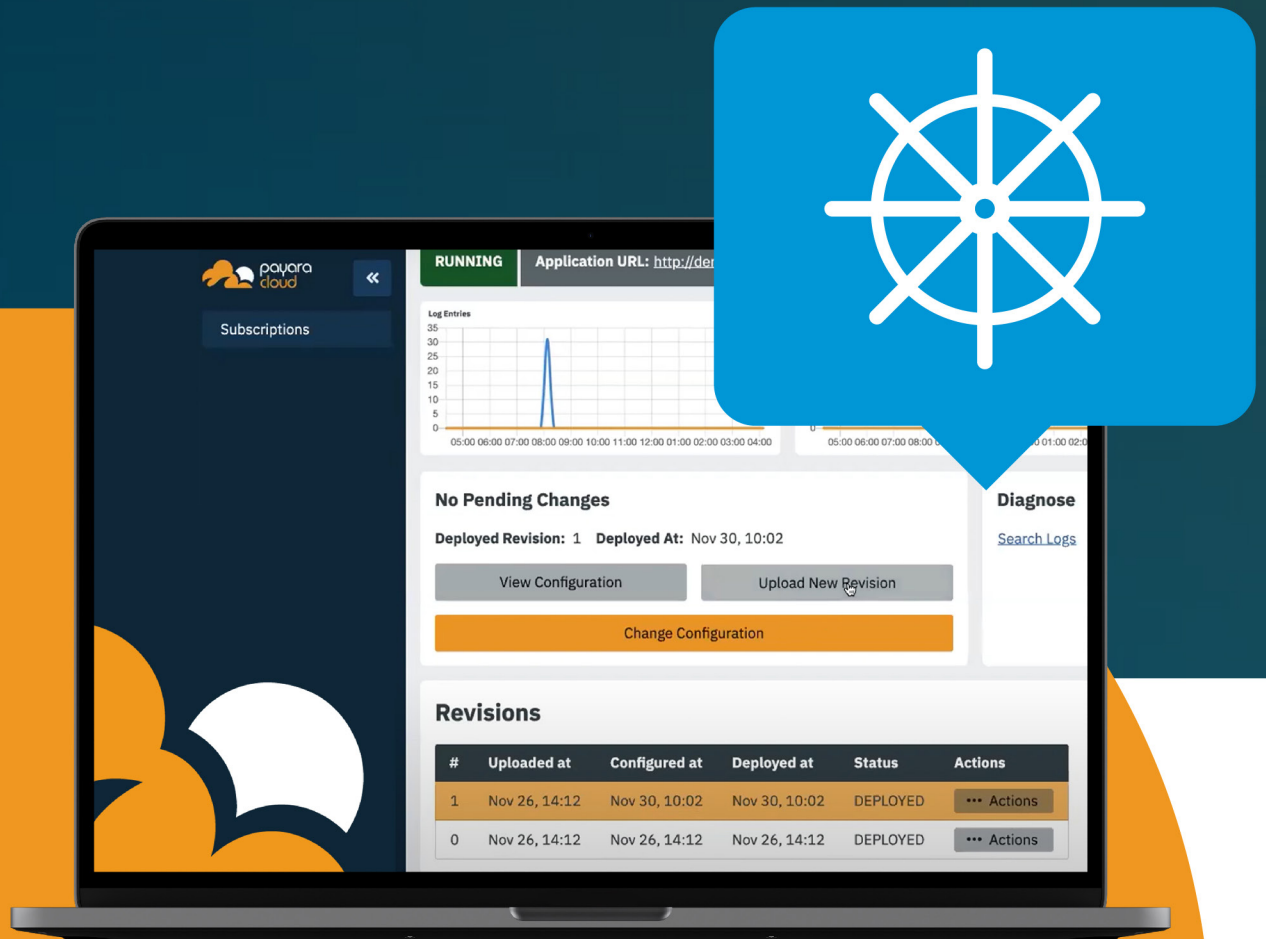




Learning Kubernetes Taking Too Long?

Automatically Deploy Jakarta
EE Apps in the Cloud



Payara Cloud - Click Deploy and Run on the Cloud.

eBook

Contents

01

*What Exactly is
Cloud-Native?*

02

Going to the Cloud

04

*The Application-
Focused Jakarta EE
Model*

05

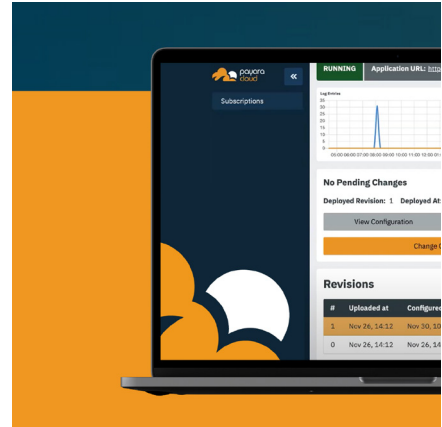
*DevOps: Developers
with Increasing
Responsibilities*

06

*Taking the Jakarta
EE Deployment Model
into the Cloud Era with
Payara Cloud*

09

*Run Your Applications in
the Cloud with Ease*



What Exactly is Cloud-Native?

The cloud-native topic seems to be one of the most frequently discussed topics over the last few years. Adapting this methodology can be challenging - but first, we need to agree on a definition for a cloud-native environment, as there are many definitions of “cloud-native” in use.

Just using the resources of a cloud provider to lift and shift existing applications to the cloud, for example, is probably a bit too simplistic as a definition for “cloud-native”.

Most people see the usage of containers and the dynamic and automated orchestration of them as the most important aspect of a cloud-native approach.

So, is cloud-native all about managing your environment and how you bring your application to the production environment to serve user requests? Some definitions of cloud-native also talk about the usage of microservices, but not

everyone agrees that this is an essential part of the cloud-native approach.

Is it possible to have a cloud-native approach with a classic Java Enterprise application, since the methodology is about operations? Or is the microservices part that some definitions mention, very important and therefore it is not possible for a Jakarta EE application to be cloud-native?

For the purposes of this document, cloud-native refers to the process of bringing your application to the production environment to serve user requests.

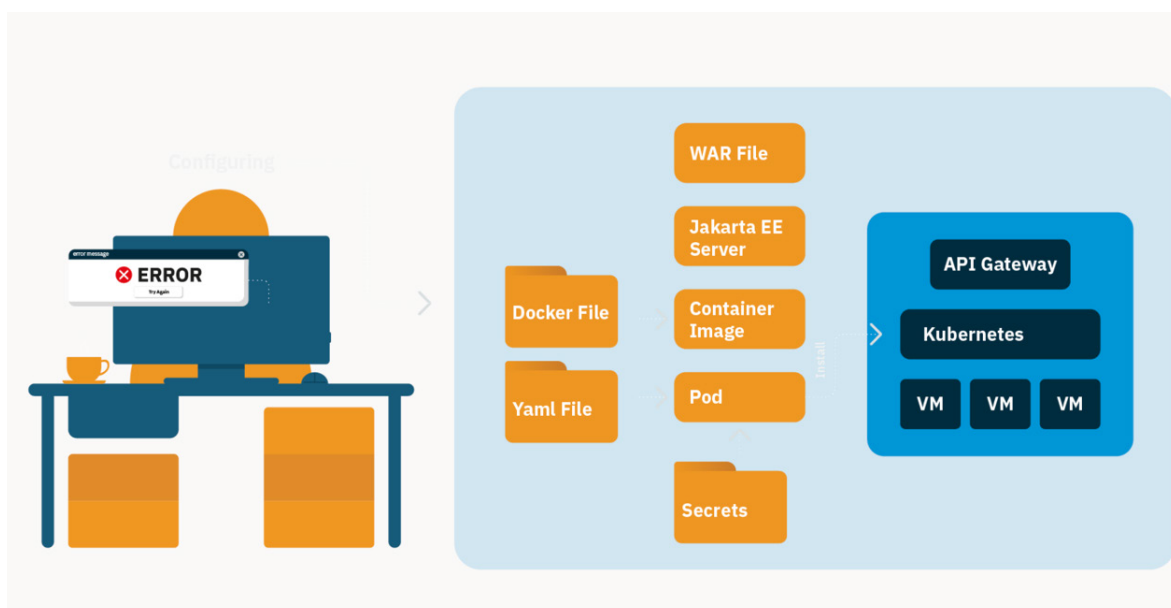
payara.cloud

Going to the Cloud

Bringing your application to a cloud environment requires more work for the initial set up and more work to maintain than running your application in an on-premises environment. Let us have a look at the main tasks of bringing your application into a cloud environment:

Most cloud environments are based on a Kubernetes cluster to run the various parts of the application. Using a containerized process offers an improvement of the hardware resources in respect to Virtual Machines. The downside of using a Kubernetes cluster is learning Kubernetes itself! Learning how to use the Kubernetes commands and the best practices around containerized images is challenging for most developers. You provide the application and the configuration to Kubernetes through YAML files, and they have challenges due to the indentation requirements. And, especially when you are running your application in a public cloud, you must take into consideration all the security-related aspects of creating container images, as other containers can access and manipulate your container if you don't carefully construct the container image.

Networking is another challenging aspect of cloud environments. Each container gets an IP address assigned so communication to and from the application is possible. This is not different from a bare-metal solution. The challenges come with the fact that you have



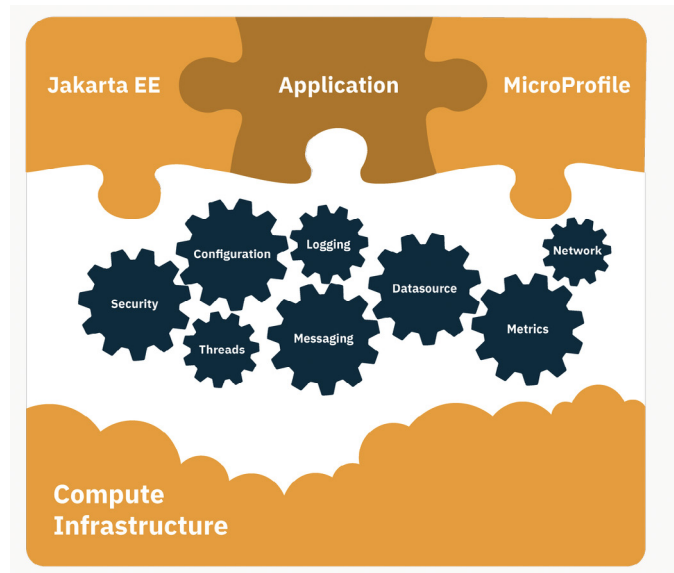
various levels and different visibility. A Kubernetes Pod can have multiple containers that can lead to port conflicts at that level. On the other side, you need to configure proper routing from the 'outside world' to your process. This also involves defining the domain name for your application and supplying a custom SSL certificate for the secure connection. Of all the infrastructure areas, networking is the least known by developers and often gives them difficulty. As a work-around solution, some less strict rules are implemented as that is the easiest solution. Opening as much as possible makes it work – but this is the worst solution when it comes to maintaining security.

Another major aspect of going to the cloud is the monitoring of the entire system. Most of the time, when companies make the move to the cloud, they also start splitting up their application into several parts. Turning it into a microservices architecture makes sense as you can leverage the benefits of Kubernetes even more. You can scale the various parts of your application individually and upgrade and install them separately. But besides the pitfall that you can end up with a distributed monolith, you also have the issue of monitoring. You need more tools to monitor all those microservices but you need a centralized place where you can check logging and resource usage. There are several standards around monitoring at the Cloud Native Computing Foundation (CNCF). There are several applications following these standards, but cloud providers each still have their own custom solutions. Integrating your applications with them can be a challenge and, once you integrate with a specific cloud provider monitoring solution, it ties you to that provider (vendor lock-in) and becomes difficult to move to another provider.

The Application-Focused Jakarta EE Model

The previous section described several aspects related to cloud and are mainly centered around the operations involved with getting your application to run on the cloud. But the main goal of our application is that we can provide business logic to our end-users. In the end, users don't care how the application is deployed and only really care about its functionality.

Java EE, and now Jakarta EE, makes it possible for developers to focus on developing the application itself, so you can solve the business logic without needing to think about infrastructure and operations.



The application server is installed, configured, and supported by the operations team. They do the setup of the environment and prepare the connection to the database, for example. The name for this connection, which is a JNDI name in your environment, just needs to be referenced in some configuration or descriptor files of your application. This has the added advantage that your application can be deployed unaltered in many environments, like test and production, as the application server configuration can point to the correct database in the test or production environment.

Another aspect of how Jakarta EE allows focusing on the application logic is the integrated specifications, already built-into Jakarta EE. With the Web profile, for providing Web-based clients like REST endpoints and UI-based interfaces with Jakarta Faces, and in the Full profile, which also supports other communications like the usage of Messaging protocols, Jakarta EE offers a set of functionalities from different areas that are ready-to-use. Developers don't need to add more libraries to provide some functionality and then deal with the difficulties of integrating them since everything is already in place - so the developer can simply focus on solving the business challenges.

DevOps: Developers with Increasing Responsibilities

With the DevOps movement, the focus of the developer is again broader than just providing a solution for the business requirements. Developers are now also involved in setting up and maintaining the infrastructure. In many cases, this makes sense as the organization is not large enough to justify the different 'silos' and in many cases improves the speed and experience of moving an application to production as there is better communication.

Instead of having different groups of people, each responsible for only a part of the entire process, it is much more efficient that you work as a single team on the problem. Developing the application and putting it on a production server is a joint task for many people.

This means that developers are confronted more often with infrastructure-related tasks and aspects. They need to learn more about the frameworks and infrastructure concepts. Using a cloud environment brings a whole new set of things to learn, like Kubernetes, routing, and setting up secure communication - to name just a few.

It can also lead to a situation where this infrastructure finds its way into the application itself, making it difficult to change later on as the infrastructure becomes part of the application. If you do not just refer to available resources, as you can do with the database connection we mentioned earlier, you make a direct and hard-coded dependency on some infrastructure component. Then you cannot easily change to another system and you need to perform much more maintenance as you probably need to rework the integration when you need to upgrade to a newer version of the integration.

It also sparked the creation of an entirely new ecosystem, the Infrastructure as Code (IaC). Since developers are good at writing applications, they approached the new challenge, dealing with the infrastructure, in a way they are comfortable with, writing code. But the environments can already be managed using command-line tools. So, the IaC needs to be developed and maintained in addition to the CLI tools. Besides this additional work, it is also something that needs to be learned by the developer. Probably in this case, it is better to invest more time in your Command Line tool knowledge than learning something entirely new - but there is still an easier way to get your web applications deployed to the cloud.

Taking the Jakarta EE Deployment Model into the Cloud Era with Payara Cloud

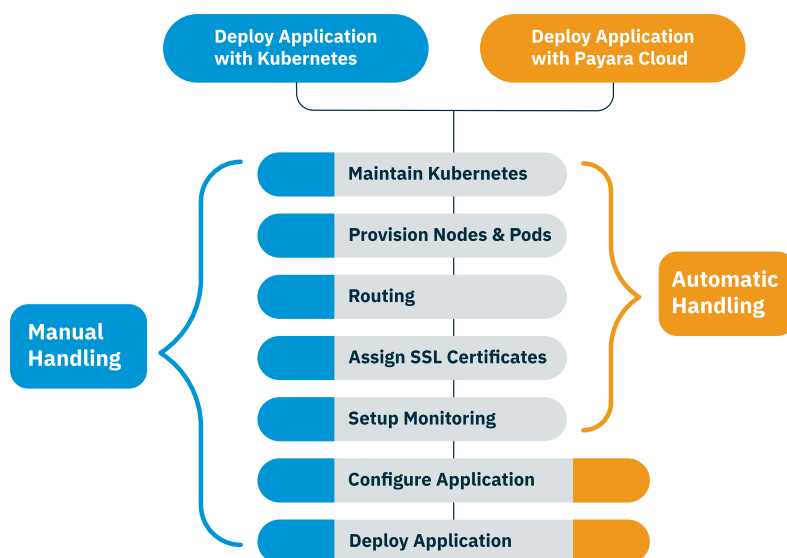
Let us go back to the initial idea of Jakarta EE: as a developer you can concentrate on solving the business problems and you can deploy the application without worrying about the infrastructure.

This deployment model should also work in the cloud era - you should upload your application and it will run in a fully-fledged cloud environment. Until now, running your applications in a cloud environment requires the developer to do a lot of infrastructure work to set up and maintain Kubernetes, provision nodes and pods, routing, assign SSL Certificates, set up the monitoring and configure the application.

Payara Cloud is a serverless architecture that allows you to build and run Jakarta EE Web apps without dealing with infrastructure management. 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.

The provisioning of the Kubernetes resources, setting up the routing, networking aspects, and providing the SSL certificate for your endpoints are all handled for you. It brings the serverless architecture idea to Jakarta EE, you just need to deploy and configure the application.

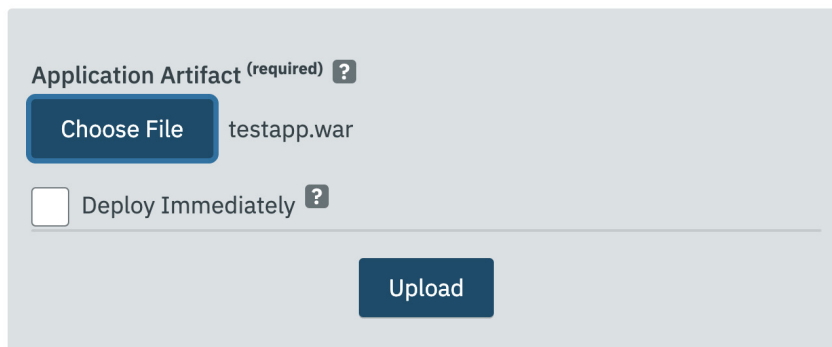
Some Platform As a Service solutions come close to this concept, but with Payara Cloud, the entire infrastructure is shielded away from the user.



The diagram on the left shows you the difference in the number of steps for deploying an application with Kubernetes when you handle each step yourself compared to the automated process of deploying an application using Payara Cloud. With Payara Cloud, you only need to configure and deploy the Jakarta EE application. All the other steps including monitoring, assigning SSL Certificates, routing, provisioning nodes and pods and even maintaining Kubernetes are handled for you by the Payara Cloud environment.

Let us go over the few simple steps using Payara Cloud to deploy an application that uses a database. The first step is to upload the

Upload Application



The screenshot shows a web form for uploading an application artifact. It has a light gray background. At the top, it says "Application Artifact (required) ?". Below this is a "Choose File" button in a dark blue box, followed by the text "testapp.war". Underneath is a checkbox labeled "Deploy Immediately ?". At the bottom right of the form is a dark blue "Upload" button.

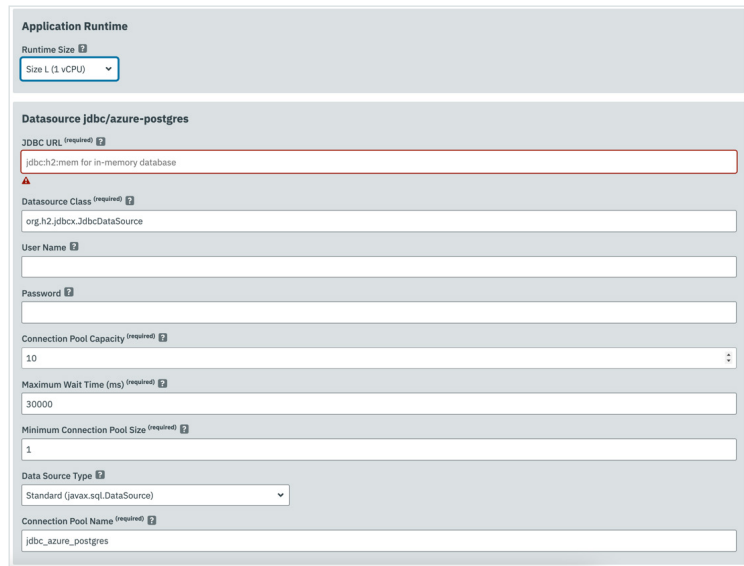
Web application into the Payara Cloud Management Console . Any Web application following the Jakarta Web profile specification can be uploaded and it will run in a containerized Payara Micro instance.

In a situation where no configuration values are needed, you can even deploy your application immediately. In our case, we need a connection to the database and have a `persistence.xml` file that defines a persistence unit within the uploaded WAR file.

The Payara Cloud deployer detects this and provides you a screen where you can enter the required values for the connection of the `jdbc/azure-postgress` DataSource in our example.

You can also supply the container size your application is running in. This depends on the memory requirements of your application and the expected load.

Payara Cloud can now start its work and make the application available for the end-users.



The screenshot shows the 'Application Runtime' configuration page. At the top, 'Runtime Size' is set to 'Size L (1 vCPU)'. Below, the 'Datasource jdbc/azure-postgres' section is expanded. It contains several fields: 'JDBC URL' with the value 'jdbc:h2:mem for in-memory database', 'Datasource Class' set to 'org.h2.jdbcx.JdbcDataSource', 'User Name' and 'Password' fields, 'Connection Pool Capacity' set to '10', 'Maximum Wait Time (ms)' set to '30000', 'Minimum Connection Pool Size' set to '1', 'Data Source Type' set to 'Standard (javax.sql.DataSource)', and 'Connection Pool Name' set to 'jdbc_azure_postgres'.

The Steps Payara Cloud Performs Are:

- 1** Prepare a container with your application.
- 2** Define a Kubernetes Deployment to run the container and application.
- 3** Provide a publicly available DNS name to access your application.
- 4** Request an SSL certificate for your application to access it securely.
- 5** Integrate with the Microsoft Azure Monitoring and Analytics system.

With Payara Cloud, you can truly focus on the business logic of your application while still benefiting from the state-of-the-art cloud technologies and cloud-native approach.

Run Your Applications in the Cloud with Ease

For an end-user, the provided functionality of an application is the most important aspect. Traditionally, Jakarta EE developers could focus on the most important aspect of development, which is the business logic that supplies the functionality of the finished application, while keeping the operations and deployments separated from the development task.

With the rise of the DevOps movement, there was more and better communications between the developers and the operations department. But it also meant that infrastructure elements were introduced into the development process and that diverts the focus of the developers away from application development. Cloud environments and the cloud-native approach also require that people learn new frameworks and processes, further distracting developers from coding - as most of the new frameworks and processes that need to be learned are not really focused on the actual goal of giving the end-users with the required functionality in the finished application.

But now with Payara Cloud, we bring the Jakarta EE deployment model to the Cloud environment.

As a developer, you can once again focus on creating the Web application based on the Jakarta Web Profile without worrying about the infrastructure. All of the infrastructure parts are taken care of for you based on the configuration value you provide during the upload of your WAR. All the Kubernetes interactions and Cloud Provider supplied functionality is taken care of for you, too, letting you run your applications in the cloud with ease.



FREE TRIAL



sales@payara.fish



**UK: +44 800 538 5490
Intl: +1 888 239 8941**



www.payara.fish

Payara Services Ltd 2023 All Rights Reserved. Registered in England and Wales; Registration Number 09998946
Registered Office: Malvern Hills Science Park, Geraldine Road, Malvern, United Kingdom, WR14 3SZ