PaaS na Prática

Computação em Nuvem para Web II
Prof. Gaio B. Oliveira

VISÃO GERAL E OBJETIVO

Ao final destas aulas, vocês terão construído e publicado uma aplicação web funcional, acessível de qualquer lugar do mundo.

OBJETIVOS

1. Distinguir modelo IaaS de PaaS.
2. Entender o que é PaaS, criar conta no Railway e provisionar a infraestrutura (servidor de aplicação e banco de dados).
3. Compreender a utilidade de plataformas em nuvem como GitHub e Railway para disponibilizar um App na web.
4. Tornar App disponível na web.

AVALIAÇÃO DA APRENDIZAGEM

Etapas para verificar o entendimento dos alunos

1. Atividade prática de criação de App e realização de deploy de maneira colaborativa.
2. Questionário conceitual.

ATIVIDADE

Guia de Aulas: Sua Primeira Aplicação na Nuvem com PHP, MySQL e Railway

Nas últimas aulas, exploramos como os serviços de nuvem funcionam, chegando a simular um ambiente AWS em nossas máquinas. Hoje, vamos dar um passo adiante e colocar uma aplicação de verdade na internet.
Vamos aprender sobre **PaaS (Plataforma como Serviço)** e usar uma ferramenta chamada **Railway**.

Aula 1: O Poder do PaaS e Nosso Ambiente no Railway

Imaginem que, para rodar nosso projeto anterior do "Mural de Recados", precisávamos instalar o PHP, o Composer, a AWS CLI e gerenciar um contêiner Docker. Isso é o que chamamos de **IaaS (Infraestrutura como Serviço)**, onde temos muito controle, mas também muita responsabilidade.
Agora, e se existisse um serviço que já nos entregasse tudo isso pronto? Um ambiente com PHP já instalado e configurado, pronto para receber nosso código?
Isso é o **PaaS (Plataforma como Serviço)**.
Analogia:

- **IaaS:** Comprar as peças e montar seu próprio computador. Você escolhe tudo, mas o trabalho é todo seu.
- **PaaS:** Comprar um notebook de última geração. Ele já vem com sistema operacional, drivers e tudo funcionando. Você só precisa instalar seus programas (nosso código) e usar.

O **Railway** é uma plataforma PaaS que faz exatamente isso. Ele cuida de toda a infraestrutura complexa para que nós possamos focar no que realmente importa: **escrever o código**.

Mão na Massa: Criando nosso Projeto no Railway

Vamos criar o ambiente para nossa aplicação de "Lista de Tarefas".
Passo 1: Crie sua conta no Railway

1. Acesse railway.app.
2. Clique em "Login" e escolha a opção "Login with GitHub". É essencial usar o GitHub, pois é assim que o Railway irá acessar nosso código para publicá-lo.
3. Siga os passos para autorizar o Railway a acessar sua conta GitHub.
4. O Railway oferece um plano gratuito que é perfeito para nossas aulas e projetos.

Passo 2: Crie um novo projeto

1. No seu painel (dashboard), clique em **"+ New Project"**.
2. Vamos começar pelo banco de dados. Selecione a opção **"Provision MySQL"**. O Railway criará um servidor de banco de dados MySQL totalmente funcional para você.

Passo 3: Adicione o serviço da nossa aplicação PHP

1. Dentro do seu projeto, clique em **"+ New"** novamente.
2. Selecione **"Empty Service"**. O Railway vai criar um "serviço" vazio, que será o ambiente onde nosso código PHP vai rodar.
3. **Renomeie este serviço:** Clique no nome gerado aleatoriamente (como "Service-...") e mude para algo como app-php.

Aula 2: Construindo a Aplicação "Lista de Tarefas" (com PDO)

Objetivo específico: Escrever o código PHP da nossa aplicação usando PDO, a forma mais moderna e segura de conectar com bancos de dados, permitindo criar, visualizar e excluir tarefas.

Preparando o Projeto Localmente

1. Crie uma nova pasta no seu computador chamada lista-tarefas-railway.
2. Dentro dela, crie um único arquivo chamado index.php.
3. Abra esta pasta no VS-Code.

Codificando a Aplicação com PDO

Vamos construir nossa aplicação passo a passo dentro do arquivo index.php.

Passo 1: A Conexão com o Banco de Dados via PDO

O Railway nos fornece as informações de conexão através de **variáveis de ambiente**. Usaremos PDO para uma conexão mais robusta.

```
<?php
// index.php

// --- 1. CONEXÃO COM O BANCO DE DADOS USANDO PDO ---
$host = getenv('MYSQLHOST');
$user = getenv('MYSQLUSER');
$pass = getenv('MYSQLPASSWORD');
$db = getenv('MYSQLDATABASE');
$port = getenv('MYSQLPORT');

// DSN (Data Source Name) - define a fonte de dados para a conexão PDO
$dsn = "mysql:host=$host;port=$port;dbname=$db;charset=utf8mb4";

try {
    // Criamos a conexão PDO dentro de um bloco try...catch
    $pdo = new PDO($dsn, $user, $pass);
    // Configuramos o PDO para lançar exceções em caso de erro
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    // Se a conexão falhar, o script é interrompido e uma mensagem de erro é exibida
    die("Falha na conexão com o banco de dados: " . $e->getMessage());
}
```

```
// --- 2. CRIAÇÃO AUTOMÁTICA DA TABELA ---
// Este código garante que a tabela 'tarefas' exista.
try {
    $pdo->exec("
        CREATE TABLE IF NOT EXISTS tarefas (
            id INT AUTO_INCREMENT PRIMARY_KEY,
            titulo VARCHAR(255) NOT NULL,
            criada_em TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ");
} catch (PDOException $e) {
    die("Erro ao criar a tabela: " . $e->getMessage());
}

// Restante do código virá aqui...
?>
```

Passo 2: Lógica para Adicionar e Excluir Tarefas com PDO

Agora, a lógica da aplicação usando *prepared statements* do PDO.

```
<?php
// ... (código de conexão anterior) ...

// --- 3. LÓGICA DA APLICAÇÃO (CRUD) ---
$erro = '';

try {
    // Lógica para ADICIONAR uma nova tarefa
    if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['add_tarefa'])) {
        if (!empty($_POST['titulo_tarefa'])) {
            $titulo = $_POST['titulo_tarefa'];
            // :titulo é um "placeholder" (parâmetro nomeado).
            // Usar prepared statements previne ataques de SQL Injection.
            $sql = "INSERT INTO tarefas (titulo) VALUES (:titulo)";
            $stmt = $pdo->prepare($sql);
            // Executa a query passando os parâmetros em um array
            $stmt->execute(['titulo' => $titulo]);

            header("Location: index.php"); // Redireciona para evitar reenvio
            exit();
        } else {
            $erro = "O título da tarefa não pode estar vazio.";
        }
```

```
    }

    // Lógica para EXCLUIR uma tarefa
    if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['delete_tarefa'])) {
        $id = $_POST['id_tarefa'];
        $sql = "DELETE FROM tarefas WHERE id = :id";
        $stmt = $pdo->prepare($sql);
        $stmt->execute(['id' => $id]);

        header("Location: index.php");
        exit();
    }
} catch (PDOException $e) {
    $erro = "Ocorreu um erro: " . $e->getMessage();
}

// --- 4. BUSCAR TODAS AS TAREFAS PARA EXIBIÇÃO ---
$stmt = $pdo->query("SELECT * FROM tarefas ORDER BY criada_em DESC");
$tarefas = $stmt->fetchAll(PDO::FETCH_ASSOC);

?>
```

Passo 3: A Interface HTML com Bootstrap

O código HTML permanece o mesmo, mas a forma de iterar sobre os resultados muda um pouco.

```
<!-- ... (código PHP anterior) ... -->

<!DOCTYPE html>
<html lang="pt-br" data-bs-theme="dark">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Tarefas | Railway</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
    <div class="container py-5" style="max-width: 600px;">
        <header class="text-center mb-4">
            <h1 class="display-5 fw-bold">📝 Lista de Tarefas</h1>
            <p class="text-body-secondary">Criada com PHP, MySQL e deploy no Railway</p>
        </header>
```

```
      <div class="card shadow-sm mb-4">
        <div class="card-body">
          <form action="index.php" method="POST">
            <div class="input-group">
              <input type="text" name="titulo_tarefa" class="form-control" placeholder="Qual
a próxima tarefa?" required>
              <button class="btn btn-primary" type="submit"
name="add_tarefa">Adicionar</button>
            </div>
            <?php if ($erro): ?>
              <div class="text-danger small mt-2"><?php echo $erro; ?></div>
            <?php endif; ?>
          </form>
        </div>
      </div>

      <div>
        <?php if (count($tarefas) > 0): ?>
          <ul class="list-group shadow-sm">
            <?php foreach($tarefas as $tarefa): ?>
              <li class="list-group-item d-flex justify-content-between align-items-center">
                <span><?php echo htmlspecialchars($tarefa['titulo']); ?></span>
                <form action="index.php" method="POST" class="ms-2">
                  <input type="hidden" name="id_tarefa" value="<?php echo $tarefa['id'];
?>">
                  <button type="submit" name="delete_tarefa" class="btn btn-sm
btn-outline-danger">✖</button>
                </form>
              </li>
            <?php endforeach; ?>
          </ul>
        <?php else: ?>
          <div class="text-center text-body-secondary p-4">
            <p>Nenhuma tarefa na lista. Adicione uma acima!</p>
          </div>
        <?php endif; ?>
      </div>
    </div>
</body>
</html>
```

Resumo da aula:
Nesta aula, escrevemos uma aplicação PHP completa usando PDO, a abordagem mais moderna e segura. Aprendemos a:

1. Conectar ao banco de dados com tratamento de exceções.
2. Criar tabelas dinamicamente.
3. Usar *prepared statements* com parâmetros nomeados para inserir e excluir dados com segurança.
4. Buscar e exibir os dados para o usuário.

Aula 3: Deploy na Nuvem!

Objetivo específico: Publicar nossa aplicação na internet através do GitHub e do Railway.

Passo 1: Crie um Repositório no GitHub

1. Vá para o GitHub e crie um novo repositório. Pode ser público ou privado. Chame-o de lista-tarefas-railway.
2. **Não** inicialize com um README ou .gitignore.
3. Siga as instruções do GitHub para "push an existing repository from the command line". Abra o terminal na pasta do seu projeto e execute os comandos:
   git init
   git add .
   git commit -m "Versão inicial da lista de tarefas"
   git branch -M main
   git remote add origin URL_DO_SEU_REPOSITORIO.git
   git push -u origin main
   Substitua URL_DO_SEU_REPOSITORIO.git pela URL que o GitHub forneceu.

Passo 2: Conecte o Repositório ao Railway

1. Volte para o seu projeto no Railway.
2. Clique no serviço que criamos para o PHP (app-php).
3. Você verá uma opção para **"Deploy from GitHub repo"**. Clique nela.
4. Selecione o repositório lista-tarefas-railway que você acabou de criar.
5. O Railway irá automaticamente detectar que é um projeto PHP e começará o processo de **build** e **deploy**.
6.

Passo 3: A Mágica do Deploy
O que o Railway está fazendo agora?

1. **Build:** Ele está lendo seu código e preparando o ambiente. Como não temos dependências (arquivo composer.json), este passo é quase instantâneo.
2. **Deploy:** Ele está colocando seu código online em um contêiner, com um servidor web (Nginx + PHP-FPM) já configurado.

Passo 4: Gere um Domínio Público

1. Após o deploy ser concluído com sucesso, vá para a aba **"Settings"** do seu serviço app-php.
2. Na seção "Domains", clique em **"Generate Domain"**.
3. O Railway criará uma URL pública para sua aplicação (algo como app-php-production-xxxx.up.railway.app).

Clique no link e... PARABÉNS! Sua aplicação está online!

Tente adicionar e excluir tarefas. Tudo o que fizemos está funcionando, agora na nuvem, acessível por qualquer pessoa.
Recapitulando a Aula 3 e o Projeto:

Resumo das aulas:

1. Entendemos o modelo PaaS e como ele acelera o desenvolvimento.
2. Provisionamos uma infraestrutura completa (App Server + DB) no Railway com poucos cliques.
3. Desenvolvemos uma aplicação PHP/MySQL segura e funcional.
4. Publicamos essa aplicação na internet usando um fluxo de trabalho profissional com Git, GitHub e deploy automático.

Este é o poder da computação em nuvem moderna. A partir daqui, as possibilidades são infinitas.