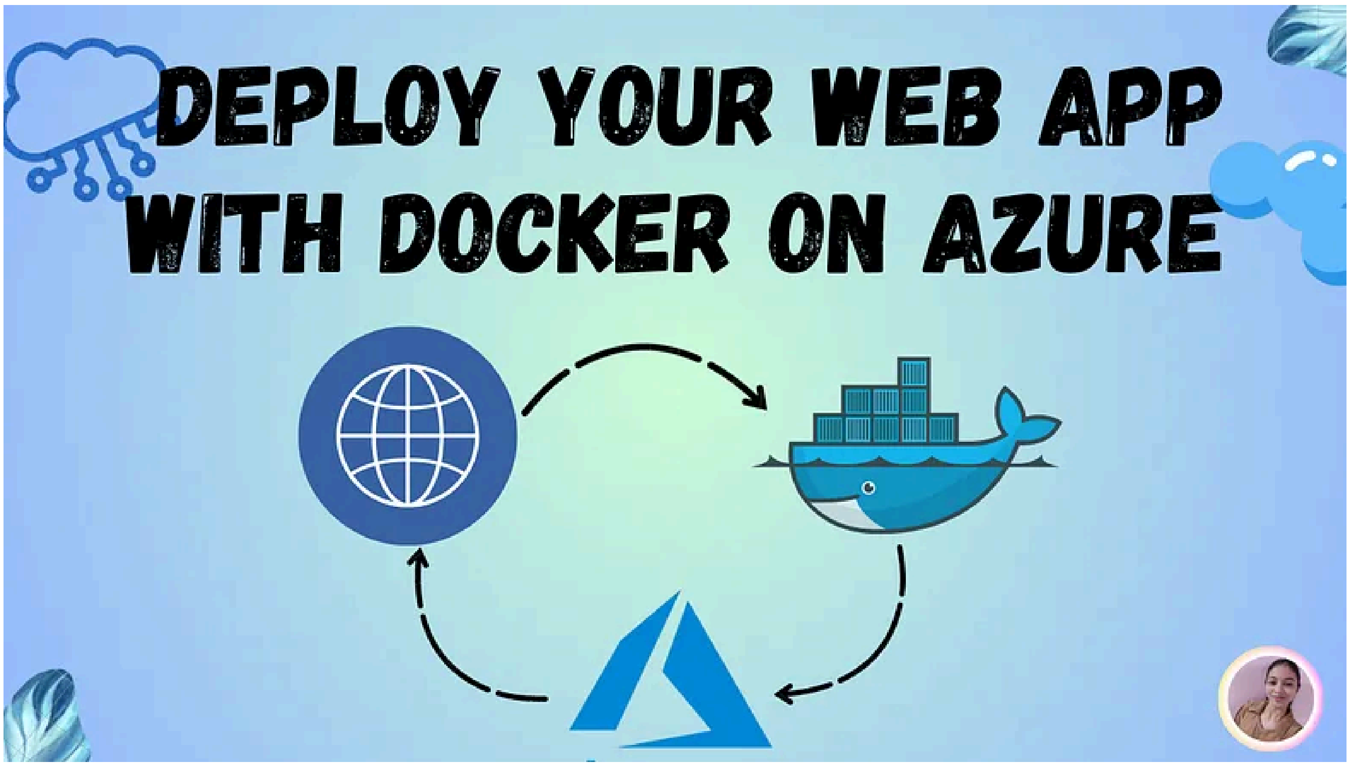
Deploying Your Web App on Azure: A Simple Guide Using Docker

Vanshita Patil · Follow
5 min read · 23 hours ago

Listen Share More



Taking your web app from local development to live on the cloud is easier than ever with Azure and Docker. Whether you're building a small personal project or preparing a large-scale enterprise application, deploying your app to Azure can be accomplished in just a few straightforward steps. In this guide, we'll walk you through the entire process — from containerizing your web app with Docker to deploying it on Azure using both the Azure CLI and the Azure Portal. Let's get your app online in no time!

Prerequisites

Before you start, make sure you have the following:

1. **Azure CLI:** Installed on your machine. Installation Guide
2. **Docker Desktop:** Installed and running. Installation Guide
3. **Azure Account:** Create one at azure.com if you don't have it.
4. **Your Web App:** A basic web application project ready to deploy.

Step 1: Add a Dockerfile to Your Project

Before you proceed, make sure you have your existing web app project structured properly. Below is a sample project structure for a Python web application, but it can be adapted for other languages or frameworks.

Sample Project Structure

```
my-web-app/
│
├── app.py                # Main application file
├── requirements.txt      # List of dependencies
├── Dockerfile             # Dockerfile for containerization
├── static/               # Folder for static files (CSS, JS, images)
│   └── styles.css
├── templates/            # Folder for HTML templates (if applicable)
│   └── index.html
└── README.md             # Project documentation
```

Navigate to Your Project Folder:
Open your terminal or command line tool (CMD or PowerShell) and navigate to your project directory:

```
cd path/to/my-web-app
```

Step 2: Containerize Your App with Docker

1. **Create a Dockerfile:** In your project directory, create a file named Dockerfile: filename: Dockerfile

```
# Use an official Node.js runtime as a parent image
FROM node:14-alpine

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Build the React application
RUN npm run build

# Serve the application using a simple server
RUN npm install -g serve
CMD ["serve", "-s", "build"]

# Expose the port the app runs on
EXPOSE 5000
```

> *Note: The Dockerfile can vary significantly based on your web app and specific requirements. Factors such as the programming language, dependencies, build process, and configuration will all influence how you structure your Dockerfile. Ensure you customize it according to your project's needs.*

2. Build the Docker Image:

```
docker build -t mywebapp-image .
```

This command builds the Docker image of your web app and tags it as `mywebapp-image`.

> *Note: Replace `mywebapp-image` with your preferred image name based on your web app and project requirements.*

Explore the Full Project on GitHub

If you need more clarity on how to structure your project or how the Dockerfile fits in, you can explore my project on GitHub for a complete reference:
My Web App Project on GitHub

Step 3: Push Your Docker Image to Azure Container Registry

Using Azure CLI

1. **Log in to Azure:**

```
az login
```

> ***Note:*** *You can run the `az` commands in any command-line tool, including Command Prompt, PowerShell, Windows Terminal, macOS Terminal, or Linux Terminal.*

2. Create Azure Container Registry:

```
az acr create --resource-group <your-resource-group-name> --name <your-registry
```

3. Login to ACR:

```
az acr login --name <your-registry-name>
```

4. Tag Your Image:

```
docker tag mywebapp-image <your-registry-name>.azurecr.io/mywebapp-image
```

5. Push the Image:

```
docker push <your-registry-name>.azurecr.io/mywebapp-image
```

Using Azure Portal(UI Option)

1. **Log in to the Azure Portal:**

- Go to portal.azure.com.

2. Create a Resource Group:

- Click on **Resource groups** > **Add.**
- Fill in the name and region, then click **Review + create** > **Create**.

3. Create Azure Container Registry:

- Search for **Container registries** in the search bar.
- Click **Create.**
- Fill in the details (name, resource group, SKU) and click **Review + create** > **Create.**

4. Login to ACR:

- Navigate to the container registry and click on **Access keys** to see the login server.

5. Push Your Docker Image:

- Use the **Cloud Shell** in the Azure Portal to run the Docker commands mentioned above.

Step 4: Create and Configure Your Azure Web App

Now, you need to create an Azure Web App and configure it to use the Docker image you just uploaded.

Using Azure CLI

1. **Create an App Service Plan:**

```
az appservice plan create --name <your-app-service-plan-name> --resource-group
```

2. Create the Web App:

Using Azure Portal(UI Option)

1. **Search for App Services:**

- In the Azure Portal search bar, type **App Services** and select **App Services** from the search results.

2. Click on Create:

- On the **App Services** page, click on the **Create** button to start the process of creating a new app service.

3. Fill in the Details:

- Provide the following details for your web app:
- **App Name:** A unique name for your web app.
- **Subscription:** Choose your Azure subscription.
- **Resource Group:** Select an existing resource group or create a new one.
- **Runtime Stack:** Select the runtime stack appropriate for your app (e.g., Node.js, Python, etc.).
- **Region:** Select the region where you want to host your web app.

3. Set Up Docker Configuration

1. **Navigate to the Docker Tab:**

- Under the creation wizard, locate the **Docker** tab and click on it.

2. Select Azure Container Registry:

- In the **Image Source** field, select **Azure Container Registry** from the available options.

3. **Choose Your Docker Image:**

- Choose your **Registry** (the one you created earlier).
- Select your **Repository** and **Image Tag** (the Docker image you previously pushed to Azure Container Registry).

Step 5: Access Your Web App

1. **Find Your Web App URL:**

- Go to the **Azure Portal.**
- Navigate to **App Services** > Select your web app.
- Copy the URL under the **Overview** section (e.g., `https://<your-webapp-name>.azurewebsites.net`).

2. **Access Your Web App:**

- Paste the URL into your browser and verify your app is running
- After deployment, navigate to:

```
https://<your-webapp-name>.azurewebsites.net
```

Conclusion

Congratulations! You've successfully containerized your web app and deployed it on Azure using both CLI and Portal methods. This guide empowers you to take your app from code to cloud with ease. Enjoy showcasing your web app!

Azure Docker Deployment Azure Web App Deployment

Follow

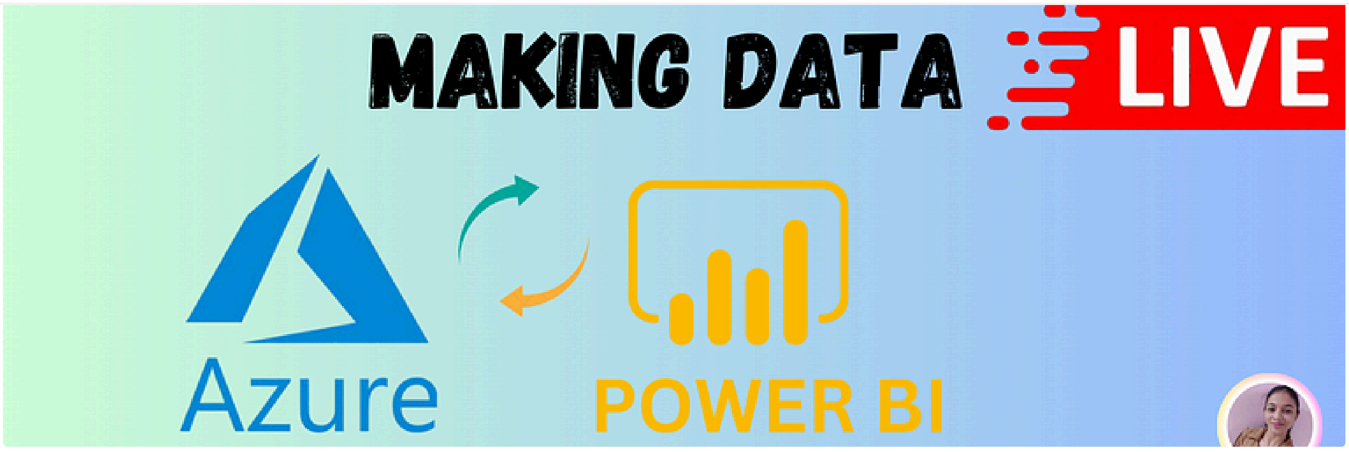
Written by Vanshita Patil

17 Followers

More from Vanshita Patil

Open in app ↗

Medium Search



Vanshita Patil

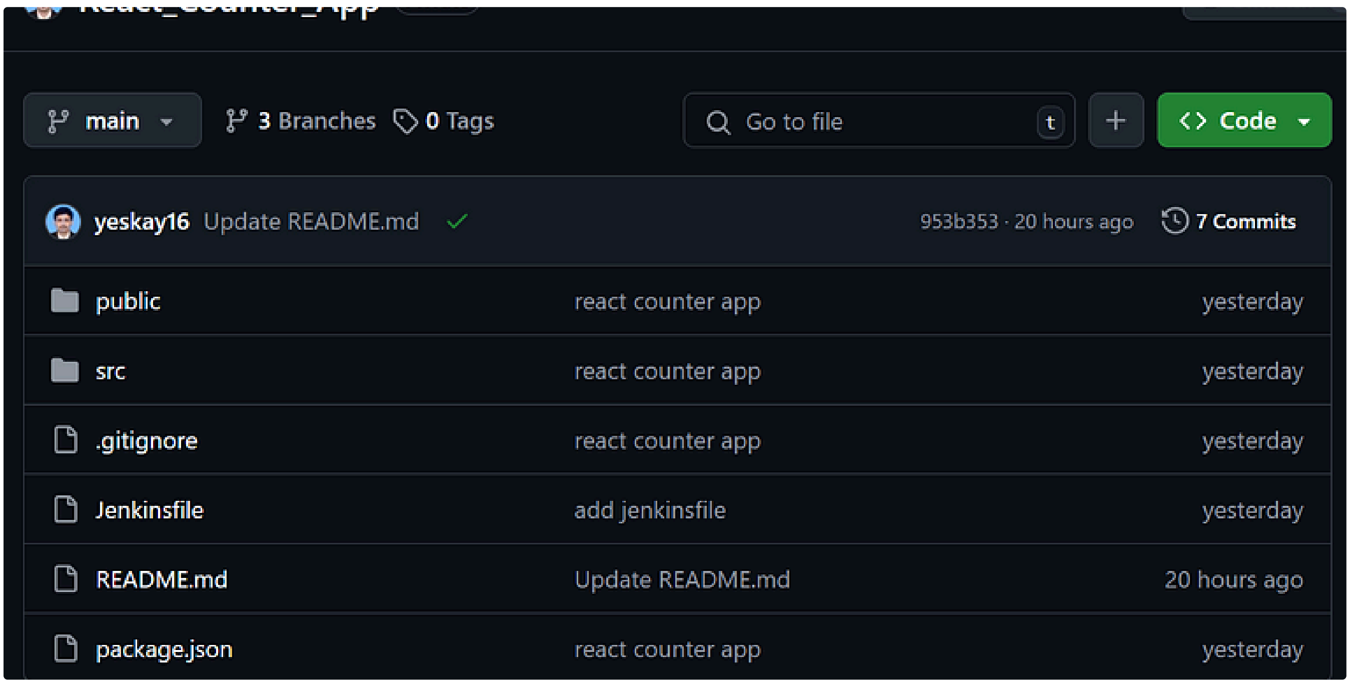
Optimizing Real-Time Data Flow: A Case Study on Integrating Azure Storage with Power BI for...

Introduction

6d ago

See all from Vanshita Patil

Recommended from Medium



yeskay16 Update README.md 953b353 · 20 hours ago 7 Commits

public	react counter app	yesterday
src	react counter app	yesterday
.gitignore	react counter app	yesterday
Jenkinsfile	add jenkinsfile	yesterday
README.md	Update README.md	20 hours ago
package.json	react counter app	yesterday

Karthik Seenuvasan

Trigger Jenkins—Multibranch Pipleline using Jenkinsfile.

https://youtu.be/vQmS0cEjTyw

Sep 25 55

Sachinthana Buddhika

CI/CD Pipeline with GitHub, Jenkins and Docker

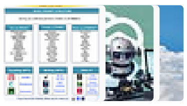
A CI/CD pipeline automates code deployment using GitHub, Jenkins, and Docker.

Jun 16 13

Lists

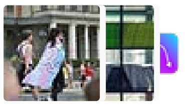
Coding & Development

11 stories · 826 saves



ChatGPT

21 stories · 817 saves

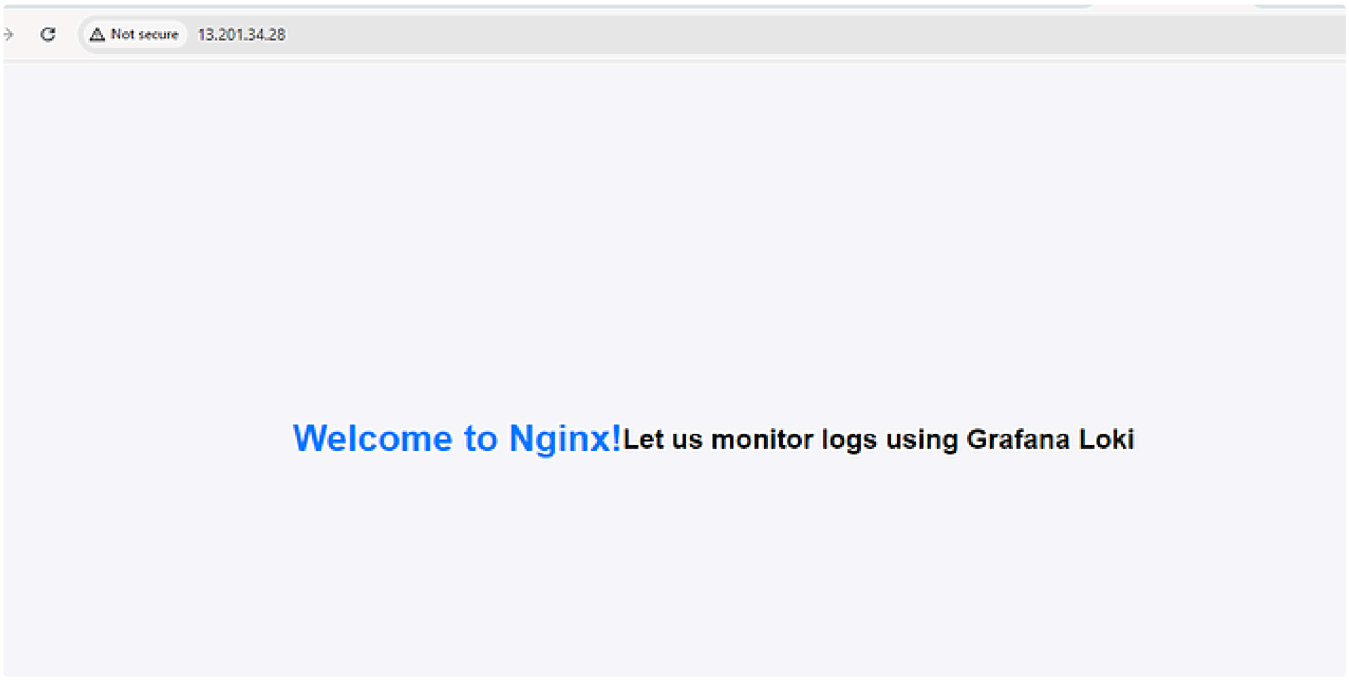


Generative AI Recommended Reading

52 stories · 1406 saves

Natural Language Processing

1733 stories · 1314 saves



Not secure 13.201.34.28

Welcome to Nginx!Let us monitor logs using Grafana Loki

Naman Sharma in DevOps.dev

Harnessing the Power of Grafana Loki: Pro Tips for Nginx Log Management

What is Grafana , Grafana-Loki and Promtail.

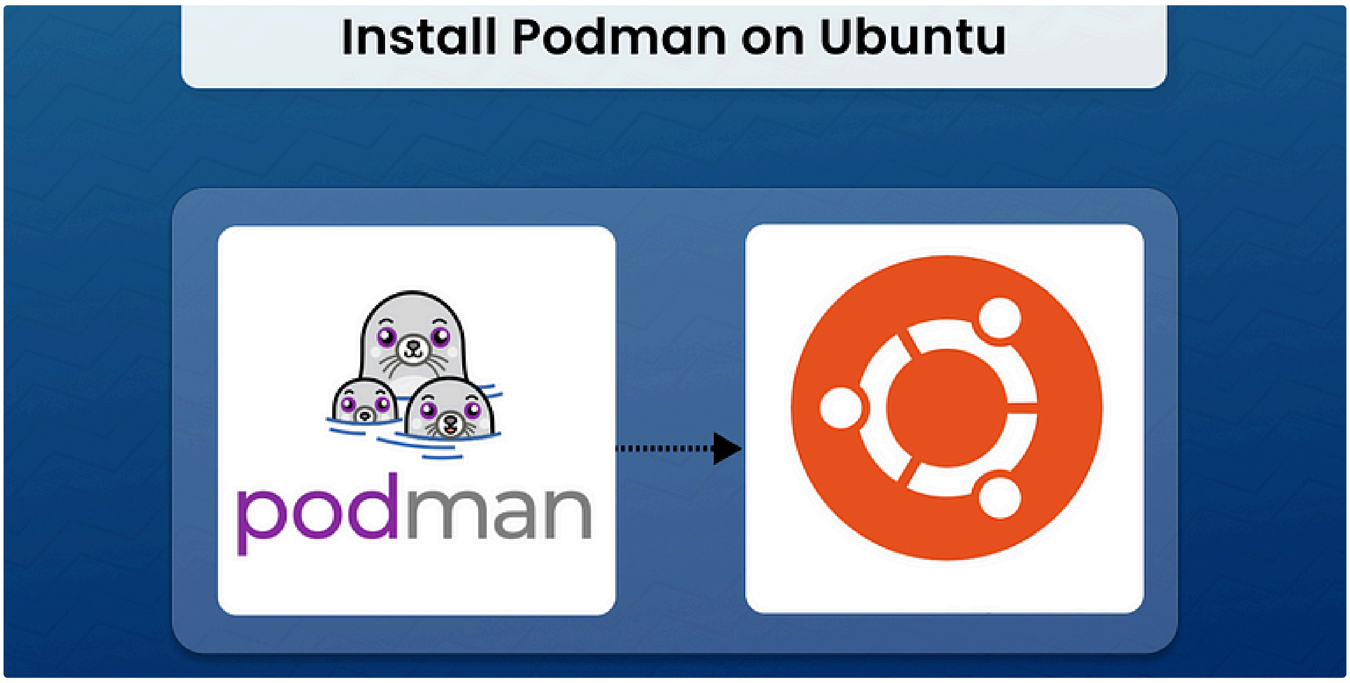
4d ago 1

Mr.PlanB

Docker, Proxmox, and VMs: The Ideal Server Setup for Maximum Control

In today's tech landscape, the demand for flexibility, efficiency, and control in server management has never been higher. As businesses...

Sep 25 52



RedSwitches

How to Install Podman on Ubuntu 20.04 and 22.04

Are you tired of the complexities of container management?

Aug 13

Harendra

Top 10 Most Used Open Source SaaS Products

Start using top open-source products for your daily tasks and save money

Sep 24 799 5

See more recommendations