

## Aim

To create a simple web application using **Python (Flask)** and deploy it on a **Public Cloud Platform (AWS Elastic Beanstalk)** to demonstrate the concept of **Platform as a Service (PaaS)**.

## Procedure

### Step 1: Create a Simple Web Application

1. Install Python and Flask on the local system.
2. Create a project folder named simple\_web\_app.
3. Create a file named app.py.
4. Write a basic Flask application that displays a welcome message.
5. Test the application locally using a web browser.

The screenshot shows the Microsoft Azure portal interface for App Services. At the top, there's a navigation bar with 'Microsoft Azure' and various search and filter options. Below it, the main content area is titled 'App Services' with a sub-section 'Future-proof all my web apps'. A message indicates 'You are viewing a new version of Browse experience. Click here to access the old experience.' The central part of the screen displays a large icon of a gear with circles and the text 'No app services to display'. Below this, there's a brief description: 'Create, build, deploy, and manage powerful web, mobile, and API apps for employees or customers using a single back-end. Build standards-based web apps and APIs using .NET, Java, Node.js, PHP, and Python.' A link 'Learn more about App Service' is provided. At the bottom, there are pagination controls ('Showing 1 - 0 of 0. Display count: auto') and a 'Give feedback' button.

### Step 2: Prepare Application for Deployment

1. Create a requirements.txt file containing the Flask dependency.
2. Ensure the application listens on the default port provided by the cloud environment.
3. Verify that the application runs successfully in the local environment.

The screenshot shows the Microsoft Azure 'Create Web App' wizard. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the breadcrumb navigation shows 'Home > App Services > Create Web App'. The main title 'Create Web App' is followed by a three-dot ellipsis. A horizontal navigation bar below the title includes tabs for 'Basics' (which is underlined), 'Database', 'Deployment', 'Networking', 'Monitor + secure', 'Tags', and 'Review + create'. A callout box in the center says 'Try Managed Instances (preview) on Azure App Service: a new option that delivers the platform benefits you rely on today, plus added features and flexibility to help you modernize applications seamlessly [Learn More](#)'. Below this, a descriptive text block states: 'App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)'. The 'Project Details' section asks to select a subscription and resource group. The 'Subscription' dropdown is set to 'Azure for Students'. The 'Resource Group' dropdown is set to '(New) Resource group' with a 'Create new' link. The 'Instance Details' section asks for a name, which is 'Web App name'. It also includes a toggle switch for 'Secure unique default hostname on' and a note 'More about this update'. The 'Publish' section has two radio buttons: 'Code' (selected) and 'Container'. At the bottom, there are buttons for 'Review + create', '< Previous', and 'Next : Database >'.

### Step 3: Create an AWS Account

1. Sign up or log in to the **AWS Management Console**.
2. Navigate to **Elastic Beanstalk**, which is a PaaS service.
3. Choose **Create Application**.

### Step 4: Deploy the Application on AWS Elastic Beanstalk

1. Select **Java** as the platform.

2. Upload the application source code (ZIP file).
3. Configure basic settings such as application name and environment.
4. Launch the environment.

The screenshot shows the Microsoft Azure 'Create Web App' wizard. At the top, there's a navigation bar with 'Home > App Services > Create Web App'. Below it, the 'Subscription' dropdown is set to 'Azure for Students' and the 'Resource Group' dropdown is set to 'resource\_group'. Under 'Instance Details', the 'Name' field contains 'resourcesapp.azurewebsites.net'. A toggle switch is turned on for 'Secure unique default hostname'. The 'Publish' section shows 'Code' selected. For 'Runtime stack', 'Java 11' is chosen. For 'Java web server stack', 'Java SE (Embedded Web Server)' is selected. The 'Operating System' is set to 'Linux'. In the 'Region' dropdown, 'Canada Central' is chosen. A note indicates 'Not finding your App Service Plan? Try a different region or select your App Service Environment.' At the bottom, there are buttons for 'Review + create', '< Previous', and 'Next : Database >'.

## Step 5: Access the Web Application

1. Wait for AWS Elastic Beanstalk to provision resources automatically.
2. Copy the generated **public URL**.
3. Open the URL in a web browser.
4. Verify that the web application runs successfully.

The screenshot shows the Microsoft Azure portal interface for creating a web app. The top navigation bar includes the Microsoft Azure logo and a search bar. Below the navigation, the breadcrumb trail shows 'Home > App Services > Create Web App'. The main title is 'Create Web App' with a '...' button. A horizontal navigation bar below the title contains tabs: Basics, Database, Deployment, Networking (which is underlined, indicating it is selected), Monitor + secure, Tags, and Review + create.

The 'Networking' section contains the following text: 'Web Apps can be provisioned with the inbound address being public to the internet or isolated to an Azure virtual network. Web Apps can also be provisioned with outbound traffic able to reach endpoints in a virtual network, be governed by network security groups or affected by virtual network routes. By default, your app is open to the internet and cannot reach into a virtual network. These aspects can also be changed after the app is provisioned.' followed by a 'Learn more' link.

Below this text are two configuration options:

- 'Enable public access \*' with radio buttons for 'On' (selected) and 'Off'.
- 'Enable virtual network integration \*' with radio buttons for 'On' (unchecked) and 'Off' (selected).

## Step 6: Demonstrate PaaS Features

1. Observe that AWS manages:
  - Server provisioning
  - Operating system
  - Scaling
  - Load balancing
2. The developer focuses only on application code, demonstrating **Platform as a Service (PaaS)**.

## Result

A simple Python-based web application was successfully created and hosted on **AWS Elastic Beanstalk**, demonstrating the working of **Platform as a Service (PaaS)**.

### Create Web App

#### Details

Subscription	c552dc0f-ec3d-4ed4-9b72-b669503ca0d1
Resource Group	resource_group
Name	resourcesapp
Secure unique default hostname	Enabled
Publish	Code
Runtime stack	Java 11
Java web server stack	Java SE (Embedded Web Server)

#### App Service Plan (New)

Name	ASP-resourcegroup-942c
Operating System	Linux
Region	Canada Central
SKU	Basic
Size	Small
ACU	100 total ACU
Memory	1.75 GB memory

#### Monitor + secure (New)

Application Insights	Enabled
Name	resourcesapp
Region	Canada Central

[Create](#)[< Previous](#)[Next >](#)[Download a template for automation](#)