RANIPPETTAI ENGINEERING COLLEGE

Name - N.DHINESH

Register No - 512121104302

PERSONAL BLOG ON IBM CLOUD STATIC WEB APPS

PHASE- 01

Department Of computer science Engineering

Defining Static Web Apps

A **Static Web Application** is any web application that can be delivered directly to an end user's browser without any server-side alteration of the HTML, CSS, or JavaScript content. While this can encompass very flat, unchanging sites like a corporate web site, static web *applications* generally refer to rich sites that utilize technologies in the browser instead of on the server to deliver dynamic content.

A Traditional Web Application

Many web applications rely on server-side generation of HTML pages to deliver a dynamic experience. The way this works, in simple terms, is as follows:

1. The user agent (browser) sends a request to a web server at a specified address.
2. The server receives the network request and sends its information along to the web application.
3. The web application connects to databases or other necessary information stores based on the request characteristics (such as the address, user session, and more).
4. The web application uses this data to dynamically generate HTML (for instance, showing a user's name in the header or populating a search page with results).
5. The server sends the created HTML to the user agent, which will then render it and display it to the end user.

Note that the web application itself may use any of a vast array of technologies, frameworks, and programming languages (Ruby, Python, Node.js, Java, .NET, and Go to name just a few). What they all have in common is that *the HTML is assembled from dynamic data on the server and sent to the browser as a complete document*.

A Static Web Application

Let's contrast this with the request cycle of a static web application:

1. The user agent (browser) sends a request to a static web server at a specified address.
2. The server receives the network request and maps the address to a "barebones" HTML file stored on the server.
3. The server sends the HTML file to the user agent, which will then render it and display it to the user.
4. The JavaScript in the HTML page uses the browser to connect to data services and fetch the information it needs to construct the content of the page.
5. The JavaScript in the page takes the data and manipulates the page's HTML, updating it with the data fetched above.

The primary difference here is that in a traditional web app, the *server* is responsible for fetching data and compiling it into HTML that the user can see, while in a static web app the *browser* is responsible for doing so.
Tip: Want to see if a given page is a static web app? Just use **View Source** in your browser. If you see user-specific data or other dynamic content in the source code, it's a traditional app. If you only see the basic structure of the page, it's a static app.

This may seem like a small difference, but it has widespread effects throughout the architecture, design, and user experience of the application as a whole. We will dive into some of the reasons you might want to build static web apps in the next chapter.

Hybrid Apps

Traditional and static apps are opposite ends of a spectrum. In between there are many ways to utilize aspects of each to get some of the benefits of both. It is common, for instance, for an application to store fully generated pages of dynamic content and then use JavaScript to fetch additional, personalized data for an individual user.

If you visit **Airbnb** while logged in, you may notice that for a second or two, the page may say "Sign Up" and "Log In". Once fully loaded, the page shows your profile

picture and name. This is an example of a hybrid app with some data rendered by the server, some by the browser.

While hybrid apps can bring the advantages of both static and traditional apps, they also come with many architectural challenges. As browser technologies and those that power static apps improve, many of the advantages of hybrid apps can now be realized in a fully static site.

DESIGN THINKING

If you’re looking for a reliable, scalable, and cost-effective web hosting solution, then migrating to a cloud web host might be the perfect choice for you. With cloud web hosting, your website is hosted on multiple servers that work together to provide maximum uptime and performance. In this article, we’ll guide you through the steps involved in migrating to a cloud web host.

Evaluating Your Current Web Hosting Solution

Before making the move to a cloud web host, it’s essential to evaluate your current hosting solution. Start by assessing your website’s traffic and resource usage. If you’re experiencing frequent downtime or slow loading times due to high traffic volumes, then it might be time to upgrade your hosting plan.

Next, consider the features and limitations of your current hosting plan. Are

you restricted by bandwidth or storage limits? Do you have access to advanced security measures like SSL certificates and firewalls? Understanding these limitations will help you choose the right cloud web hosting plan that meets your needs.

Choosing the Right Cloud Web Hosting Provider

There are many cloud web hosting providers available today, each with their own unique offerings and pricing plans. When choosing a provider, consider factors like reliability, scalability, security features, and customer support.

Look for providers that offer guaranteed uptime percentages and automatic failover mechanisms in case of server failures. Scalability is also crucial – ensure that your provider offers flexible resource allocation options so that you can easily scale up or down as needed.

Migrating Your Website to a Cloud Web Host

Now that you've chosen your cloud web host provider, it's time to migrate your website over. The process involves transferring all of your website's files, databases, and configurations to the new server.

Most cloud web hosting providers offer migration assistance as part of their service. However, if you're migrating on your own, start by backing up all of your website's files and databases. Then, upload them to the new server using an FTP client or the provider's migration tool.