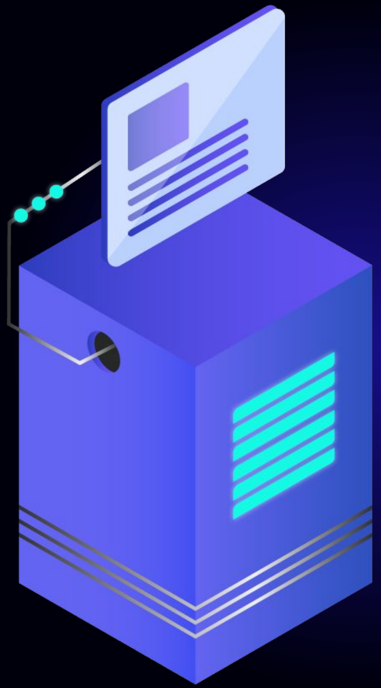
UNDERSTANDING INFRASTRUCTURE

Presented By: Abdul Hadi Waseem

TABLE OF CONTENTS



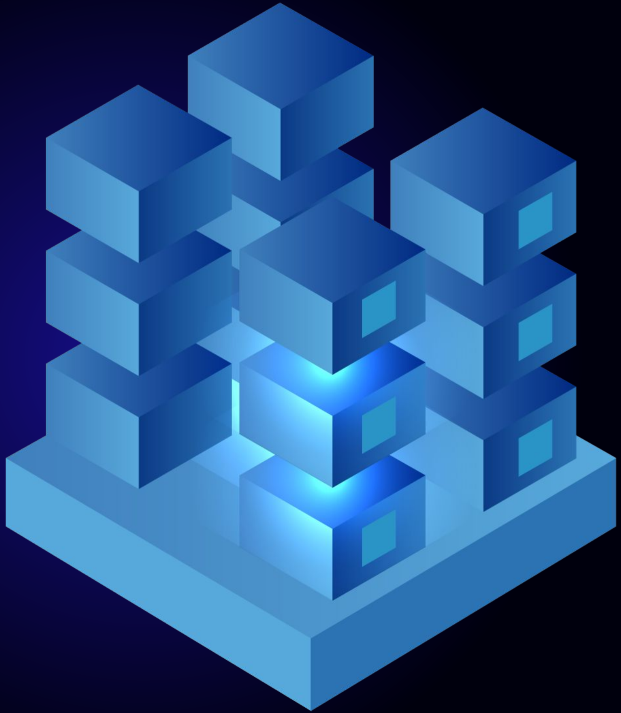
WHAT IS INFRASTRUCTURE

INFRASTRUCTURE:

- Infrastructure refers to the fundamental facilities and systems necessary for the functioning of applications
- It provides compute, network, workplace, and datacenter capabilities needed to run the applications
- Traditionally, it required large data centers, servers, and IT personnel to manage the setup , maintenance and deployment stage of an application

02

INFRASTRUCTURE AS CODE

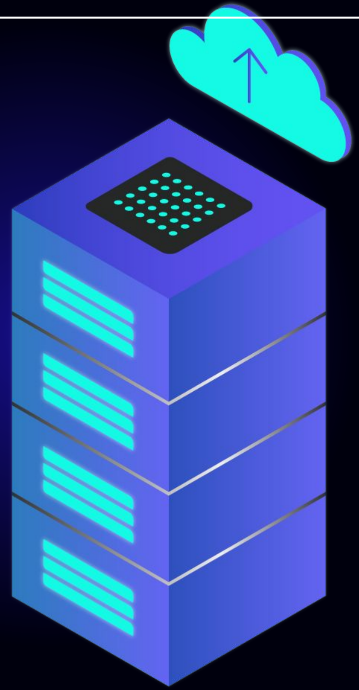


Before Automation Of Devops

- Before DevOps automation, deploying an application required a manual setup process which involved buying servers, creating networking, configuring routes, accessing individual machines, and running install scripts. There was also a maintenance phase before final deployment that involved maintaining the application, recovering from crashes, and performing other tasks that were typically performed by system administrators. This approach led to increased costs and risks of errors.
- With the growth of the web and the emergence of DevOps, Infrastructure as Code (IaC) became the preferred method for repeatable and reliable infrastructure provisioning.

After Automation Of Devops

- IaC automates tasks that were previously done manually by system administrators and teams, and different tools are available for different jobs i.e Docker and Terraform . The code in IaC is a version-controlled description of the desired infrastructure state, which can refer to networks, servers, queues, and databases. This enables developers to define their infrastructure as code, store it in a version control system, and automate the provisioning and maintenance of their infrastructure.



3 MAIN STEPS

1 Infrastructure provisioning

spinning up servers, configuring networks, creating load balancers, and other base level infrastructure tasks, including setting subnet masks

2 Configuring Of Provisioned Infrastructure

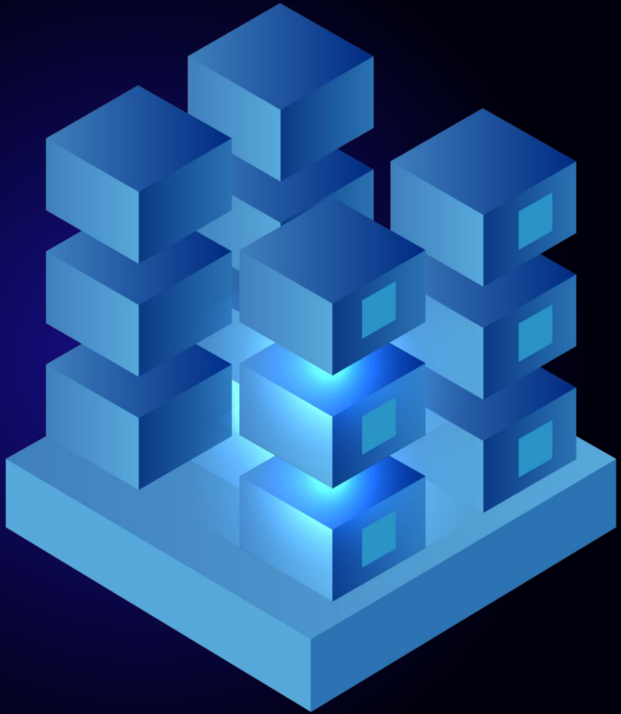
installing apps/softwares on server and managing them

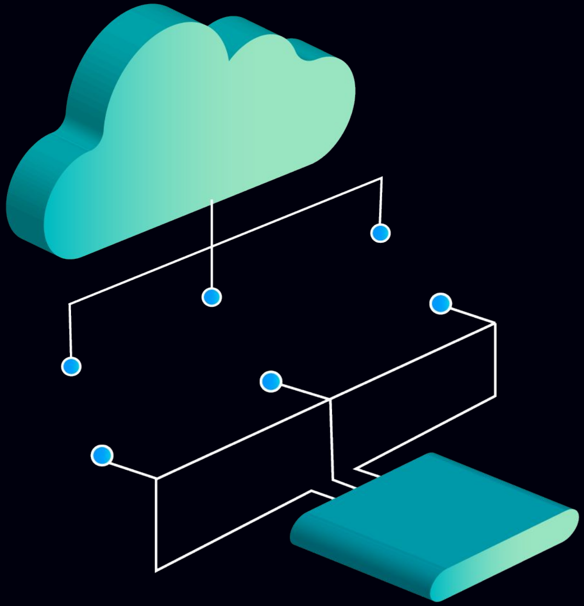
3 Final Deployment

Deploying on provisioned and configured infrastructure. But Docker combines both the second and third steps

03

FRAMEWORK DEFINED INFRASTRUCTURE (FDI)





FDI

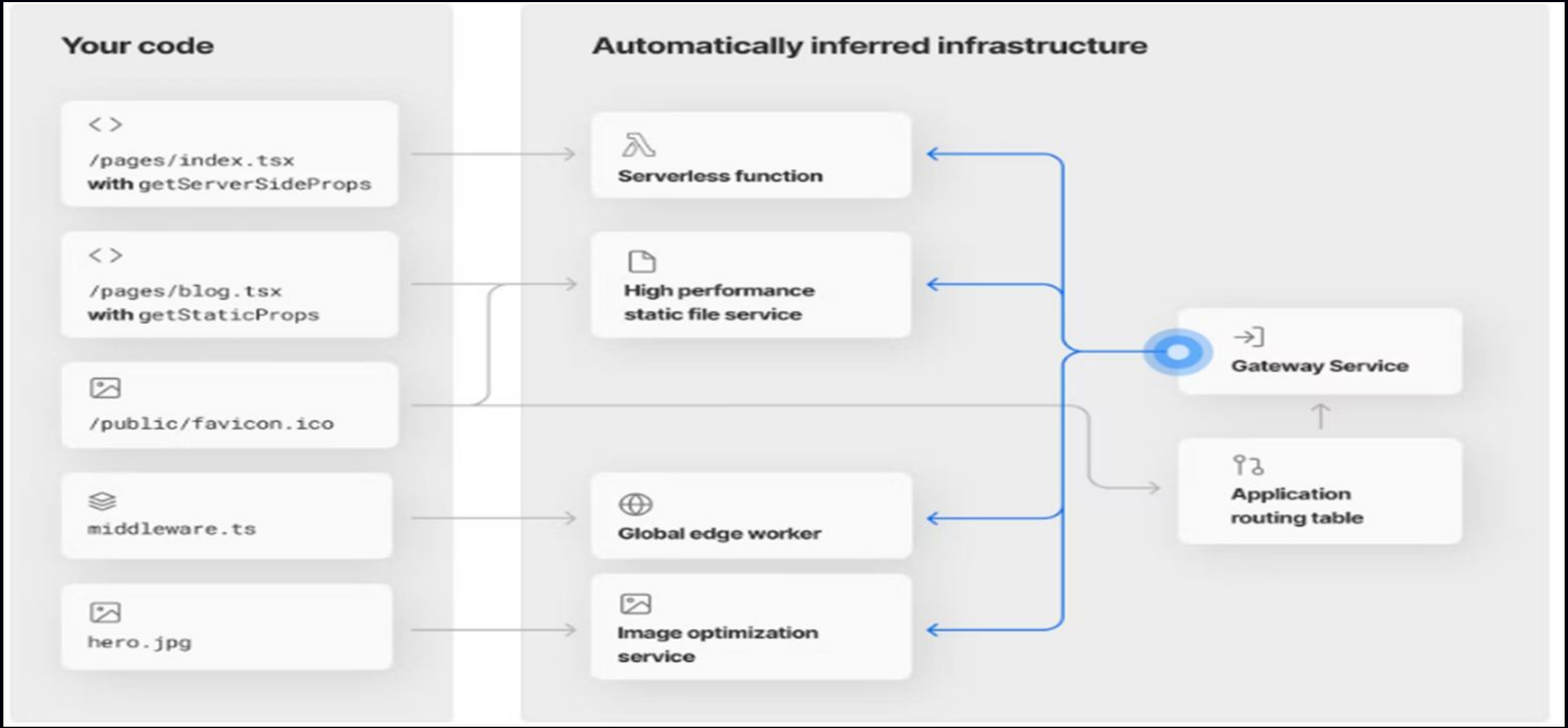
- The third step i.e deployment of application automatically provisions infrastructure derived from framework and the application written in it
- It means we don't have to setup and configure infrastructure anymore instead we just deploy and the FDL will do the rest

MORE PRECISE EXPLANATION

- Imagine if we had a program that translates the code at build time , understands it and generates IAC configurations needed to run softwares???
- This means more predictable, lower cost, and lower risk DevOps through truly serverless—dare we say, *infrastructureless*—architecture.

WHAT FDI DOES?

Framework-defined infrastructure (FDI) masks cloud primitives like servers and message queues, providing portability and simplifying infrastructure configuration. FDI prioritizes product code development over system management, emphasizes secure services, and maintains the framework's local development tools' usability. Frameworks provide structure and abstraction for applications, with FDI automatically mapping framework concepts.



The route table generated from a framework would eventually be deployed to our gateway service, which then knows how to invoke the correct infrastructure primitive for any given route.

BEHAVIOUR WITH SERVER SIDE PROPS:

```
export default function BlogPosts({ posts }) {
return posts.map(post => <BlogPost key={post.id} post={post} />)
}
export async function getServerSideProps() {
const posts = await getBlogPosts();
return {
props: { posts }}}
```

GET SERVER-SIDE PROPS:

This is a function provided by next js itself which dynamically renders the page which is holding this function ie It automatically wakes up whenever there is a view/call on page which has this function and it fetches the data and renders it on the page

NOTE: the page which has this function is completely server side and not client side

To render such a function we must require a very powerful infrastructure but fortunately it has already been provided by nestjs itself It means that the FDI in vercel's case attaches a serverless lambda function with the code which renders it

When to invoke this function? this knowledge is deployed to the gateway as part of routing table

BEHAVIOUR WITH GET STATIC PROPS:

```
export default function BlogPosts({ posts }) {
return posts.map(post => <BlogPost key={post.id} post={post} />)
}
export async function getStaticProps() {
const posts = await getBlogPosts();
return {
props: { posts }}}
```

GET STATIC-SIDE PROPS:

This means that there is no need to render page on every view,function is invoked at build time and static html page gets generated which will be delivered to user every time

If you change the getserversideprops with getstaticprops then.

Again with FDI in vercels case it is automatically inferred that no more lambda function is required and it wont get deployed with the code and the routing table gets updated

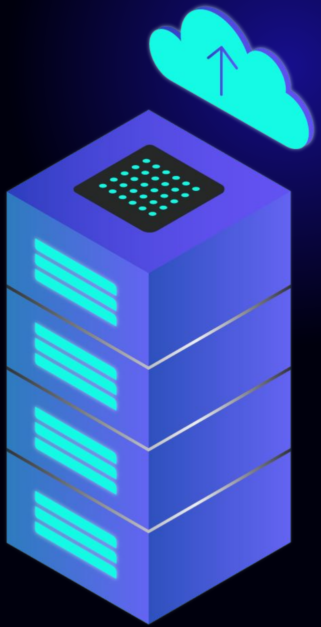
SOLVING LOCAL DEVELOPMENT PROBLEM FOR SERVERLESS

- While serverless architecture greatly simplifies managing production infrastructure, the systems that run serverless code are often complex and proprietary, which can cause problems with local development. In a worst case scenario, local development isn't possible, and developers must test local changes by making comparatively slow deploys to the production infrastructure. In a best case scenario, developers must create a local simulation of the complex serverless stack, which then has to stay up-to-date with the production environment.
- Framework-defined infrastructure completely eliminates this problem. Because framework-defined infrastructure lets the framework itself dictate the production behavior, local development can just use the framework's own tooling, while production infrastructure is automatically configured to be a scaled and optimized representation of the same underlying behavior

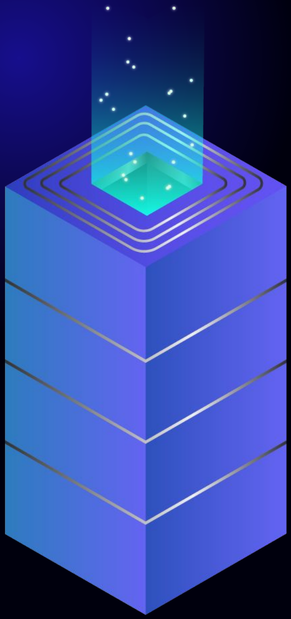
FRAMEWORK DEFINED INFRASTRUCTURE AND IMMUTABLE DEPLOYMENTS

- Immutable deployments solve this mismatch between an application's and the infrastructure’s version history. immutable deploys create a completely new infrastructure for every single deployment that is ever made. This infrastructure is then never changed—it is immutable.
- Immutable deployments are impossible to perform in a world where infrastructure is mapped to finite physical resources, which would get used up as new deployments get made. However, in a serverless world, where unused infrastructure can scale to zero, immutable deployments become possible to perform, as unused deployments do not take up physical compute resources beyond the baseline storage needed for their contents

CONCLUSION



1. Framework-defined infrastructure (FdI) is an evolution of IaC
2. Framework maps its pattern on infrastructure primitives(basic code written) so that it can be run in scalable production system
3. How immutable deployments based on serverless architecture, solves the problem of different versions of code and infrastructure making it scalable and cost effective during low usage



THANKYOU