Cloud-Native and Infrastructure as Code

This lesson explains cloud-native and infrastructure as code.

We'll cover the following

Overview

Being in the current software development universe, chances are you've heard the term *cloud native* more than once.

What does it really mean?

Cloud-native is a way of developing software that is written with the intent of being deployed on the cloud to make the most of the resources and features of the cloud infrastructure.

Building microservices and running them in containers managed by container orchestration tools is an example of a *cloud-native* application development approach.

We've discussed how microservices, containers, and orchestration tools help cut down the time involved in developing, shipping, and managing software in detail in previous lessons. The *cloud-native* approach of building software is significantly different than building traditional monolithic applications.

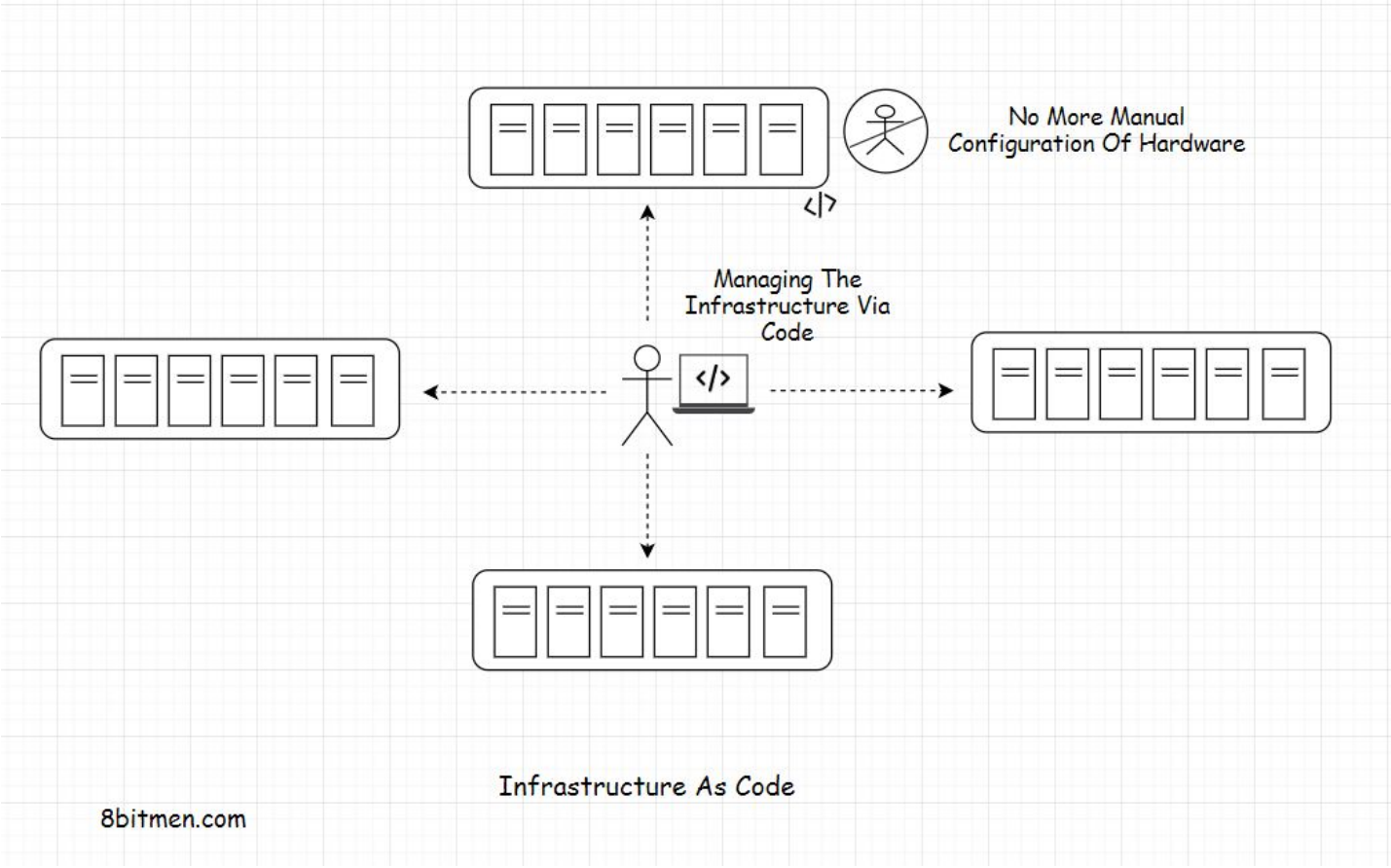
Cloud-native apps focus on keeping the components *stateless* and *loosely coupled* to facilitate better *monitoring, management, application portability, scalability*, and *high availability*. There are a plethora of plug-and-play cloud tools powered by APIs that assist the *cloud-native* application development.

Deployment pipelines are built to deliver cloud-native software at the optimum time managed by the *DevOps* teams. You'll see more on this later.

This is pretty much it regarding cloud-native. Let's move on to *infrastructure as code (IaC)*.

Infrastructure as code

Infrastructure as code, or *IaC*, means managing the physical infrastructure via code. It's a step towards automating infrastructure configuration and management as opposed to doing things manually.



Until now, to get the infrastructure prepped up to run our code, we've been configuring hardware manually or via scripts.

How does handling everything via code help?

Need for infrastructure as code

We can manually configure infrastructure for one data center when running a large-scale service. Imagine configuring and managing the infrastructure for the same service spanning across multiple data centers globally.

First off, doing things manually is prone to errors, unlike when the same process is automated. We can run an automated process a bazillion times and can be certain

that it won't throw an error. But if we do the same task a bazillion times manually, odds of us making mistakes at least a few times are pretty high.

Here are a couple of instances where human errors have resulted in a service outage.

A small configuration mistake took down half of the internet in Japan (see details here).

Human error caused a massive *AWS* outage (click here for more info).

Though it's tricky to automate everything, the infrastructure as code approach tries to cut down the manual process as much as possible. Automating things improves productivity and helps with standardizing configuration across millions of machines to facilitate consistent infrastructure behavior.

Another upside of controlling the infrastructure via code is replicating the same infrastructure configuration for creating new environments, such as *Dev, Pre-production, Staging* and so on. All this done manually would require a lot of time and again would be error-prone.

Some of the tools that enable us to achieve this are *Ansible*, *Chef*, *Puppet*, and *AWS Cloud Formation*.

With this, we have reached the end of the deployment infrastructure and technologies chapter. In the next chapter, let's talk about the code deployment workflow.

First, you'll take a quick quiz in the next lesson.