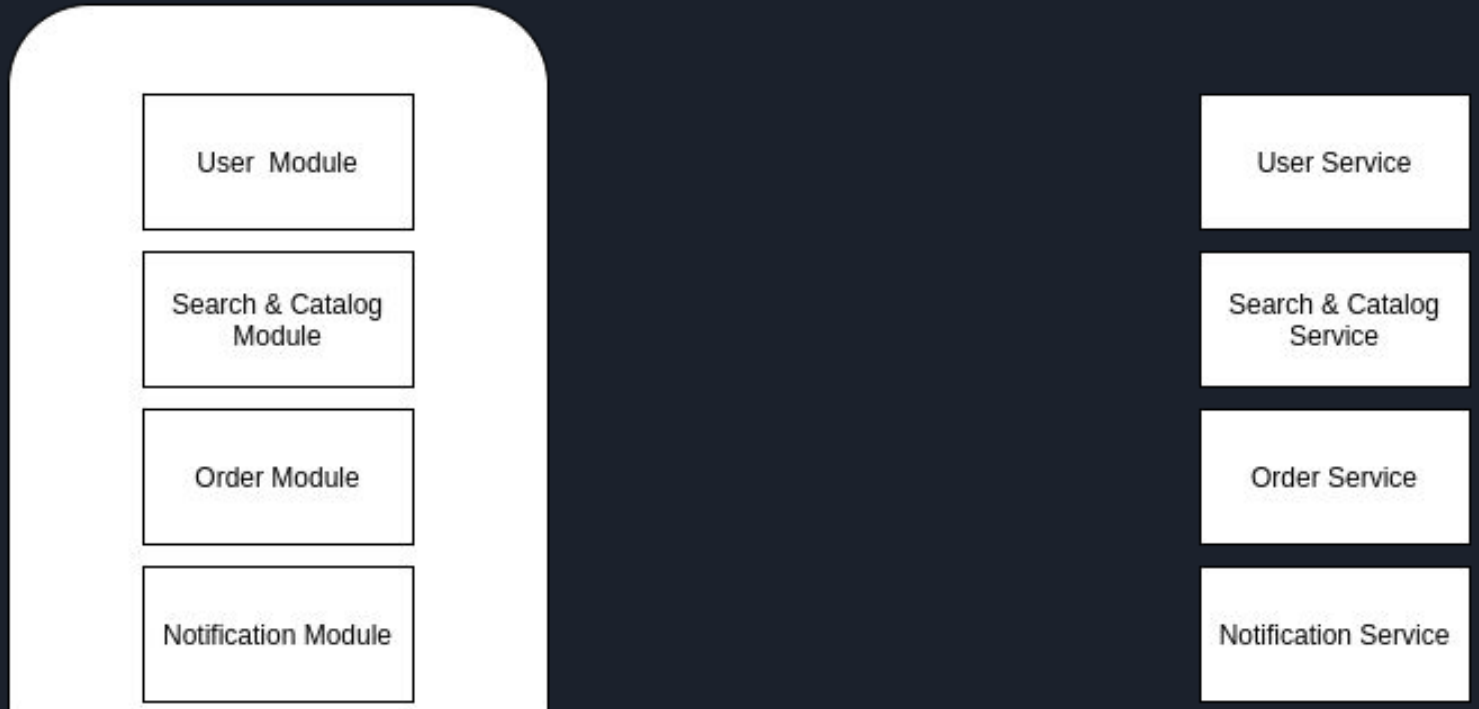
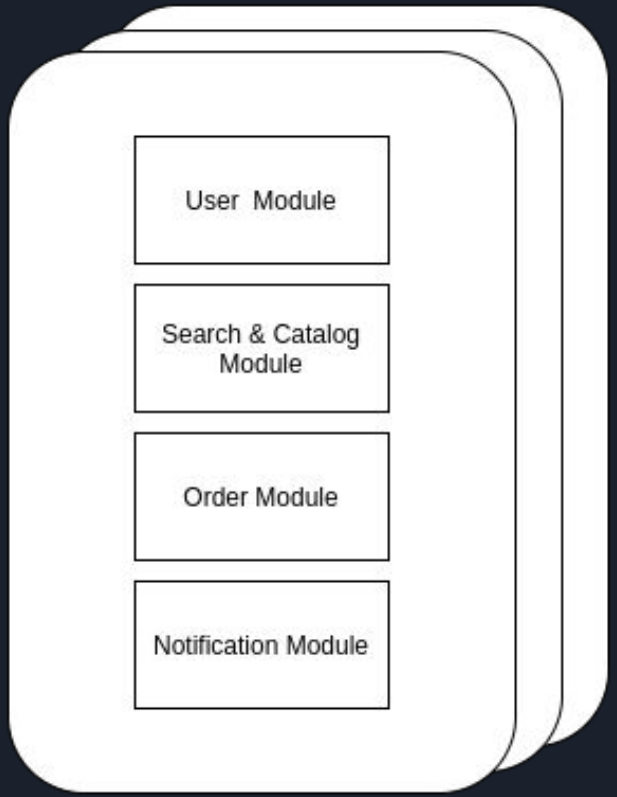
Kubernetes 101

Rounak Raj

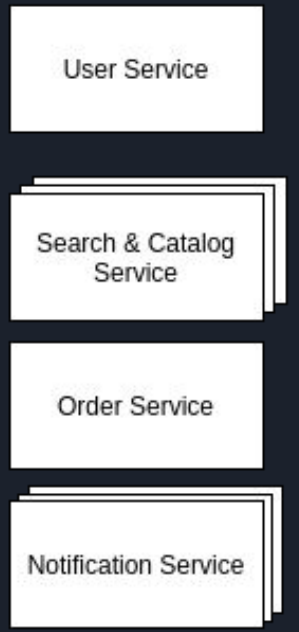
Monolithic vs Microservices



Monolithic vs Microservices - Scaling



Monolithic Application



Microservices

Monolithic vs Microservices - Service Discovery

Monolithic applications doesn’t require any service discovery as its just a single application and different modules can be called as just simple method/function calls.

Microservices needs service discovery as different modules are running as different services, so in case a service needs to communicate with other services(inter-service communication), it needs to know where to call.

Docker

Package your application with all required dependencies, so that there's no issue with dependencies wherever you deploy.

To Dockerize your application you need to write a Dockerfile.

Every time you build your application for CI/CD, you create a new Docker image, give a unique tag, and keep that image in any public/private repository. eg. DockerHub, AWS ECR, GCP GCR

Next time you need to deploy this build, you can deploy same image anywhere (DEV/TEST/QA/PROD).

A running image is called container. Like running class is called object. (Instantiated is the word)

Running Docker Images(Docker Container)

Now since you have created Docker image, you need some environment to run this.

You can run this Docker image in your own system, or any VM(AWS EC2, GCP Compute Instance), or any environment which gives you ability to run Docker images(AWS ECS, AWS EKE, GCP GKE, Azure AKS).

Running this on you own system doesn't helps as most probably your system will not be serving requests.

Running this Docker image in VMs makes sense, as we got rid of issues related to dependencies and env variables. But running in VMs will not give you any extra benefit.

Kubernetes now comes to our rescue.

What is Kubernetes(K8)?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

Features that Kubernetes provide out-of-the-box:

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

What is ! Kubernetes?

No magic. (Though there are lots of hidden stuff which might make you believe that it is)

It’s not intuitive, it’s as smart as you have configured.

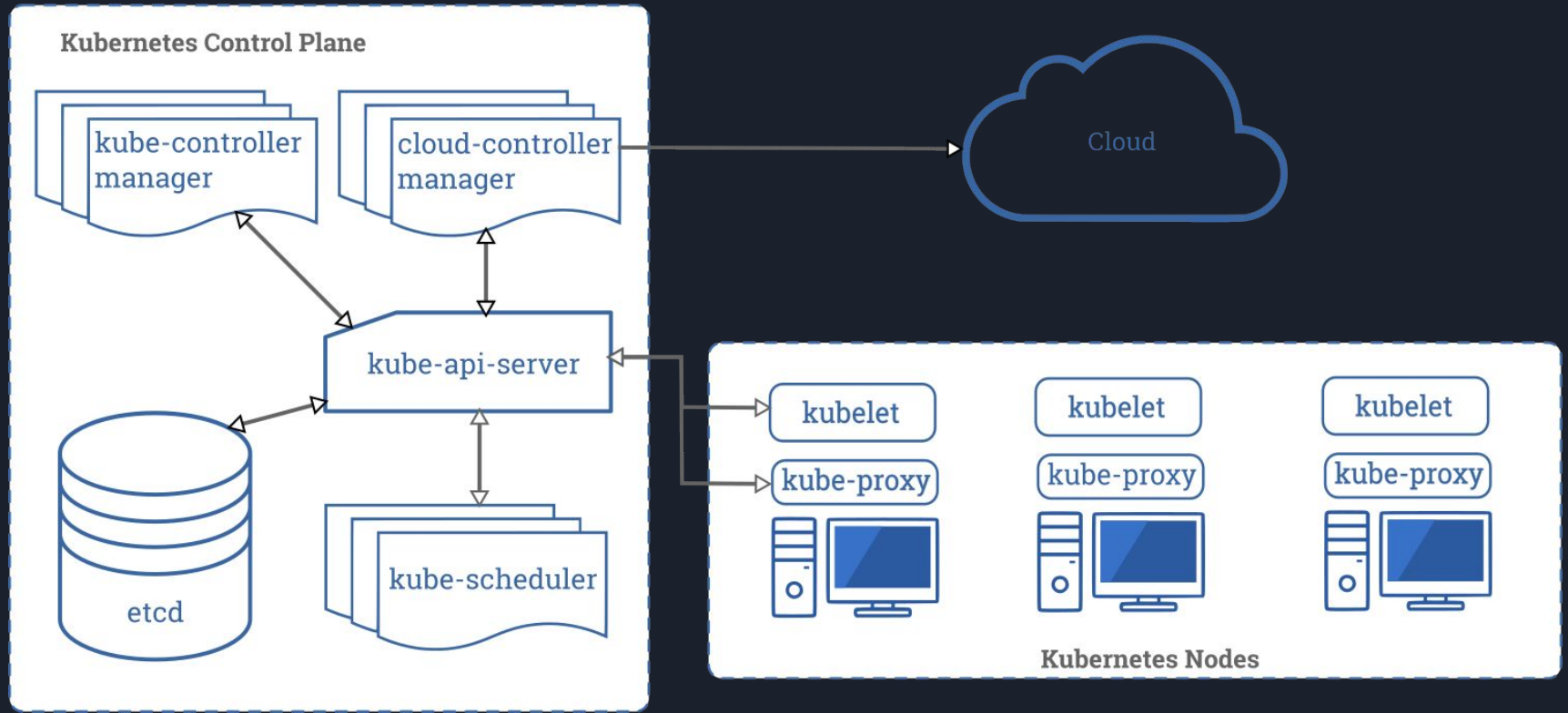
It’s not easy to learn, learning requires effort and patience.

Kubernetes Components

A Kubernetes cluster consists of the components that represent the control plane and a set of machines called nodes.

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

Kubernetes Components



Kubernetes Pods

A Pod is the basic execution unit of a Kubernetes application--the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents processes running on your cluster.

A Pod encapsulates an application's container (or, in some cases, multiple containers), storage resources, a unique network identity (IP address), as well as options that govern how the container(s) should run. A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

Docker is the most common container runtime used in a Kubernetes Pod, but Pods support other container runtimes as well.

Key

Deployment

Service