**Kubernetes Installation Options: The Hard Way, Kubedm, MiniKube, Managed K8s (EKS, AKS, OKE, GKE)**

This Blog lists different ways to **set up** and **run Kubernetes**. So when you install[**Kubernetes**](https://k21academy.com/docker-kubernetes/kubernetes-for-beginners/), choose an installation type based on ease of maintenance, security, control, available resources, and expertise required to operate and manage a cluster.

You can deploy a **Kubernetes cluster** on a local machine, or cloud, or choose a managed Kubernetes cluster. Kubernetes is composed of a minimum of 1 master node and 1 worker node. So it’s advisable to learn  [**Kubernetes Architecture**](https://k21academy.com/docker-kubernetes/kubernetes-architecture-components-overview-for-beginners/) before you start with the installation procedure.

**In this blog, I’ve Covered:**

* [**Everything from Scratch (Hard Way)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#1)
* [**Unmanaged (Installer Based)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#2)
  + [**Kubernetes with kubeadm**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#3)
  + [**Kubernetes with Kops**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#4)
  + [**Kubernetes with Kubespray**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#5)
* [**Kubernetes for Development**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#6)
  + [**Kubernetes using Minikube**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#7)
  + [**Kubernetes using Docker for Desktop**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#8)
* [**Kubernetes in Production**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#9)
* [**Managed Kubernetes (Platform Based)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#10)
  + [**Elastic Kubernetes Service (EKS)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#11)
  + [**Azure Kubernetes Service (AKS)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#12)
  + [**Google Kubernetes Engine (GKE)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#13)
  + [**Oracle Kubernetes Engine (OKE)**](https://k21academy.com/docker-kubernetes/kubernetes-installation-options/#14)

**Everything from Scratch (Kubernetes the Hard Way)**

**Kubernetes The Hard Way** is optimized for learning, which means taking the **long route** to ensure you understand each task required to bootstrap a Kubernetes cluster.  This is for someone planning to support a production Kubernetes cluster and wants to understand how everything fits together. This is **not** for people looking for a **fully automated** command to bring up a Kubernetes cluster. The guide uses the API server certificate for etcd as it uses a stacked etcd configuration. We will generate separate certificates and keys for etcd as we are running an external etcd setup.

We will place the etcd cluster behind a load balancer that gives us **multiple advantages**.

* The etcd nodes can have ephemeral IPs.
* You can add and remove etcd nodes according to your requirements.
* NGINX provides an auto health check of its back-end members, and it would not send traffic to an unhealthy etcd instance avoiding runtime issues.
* You don’t need to update the control plane configuration if you make changes to the etcd cluster (such as adding or removing etcd nodes).

We will allow only the desired traffic and block the rest of it. That is required to protect our cluster from unauthorized access. We will encrypt secrets at rest on the etcd cluster as suggested in the original guide.

**Unmanaged Kubernetes Installation (Installer Based)**

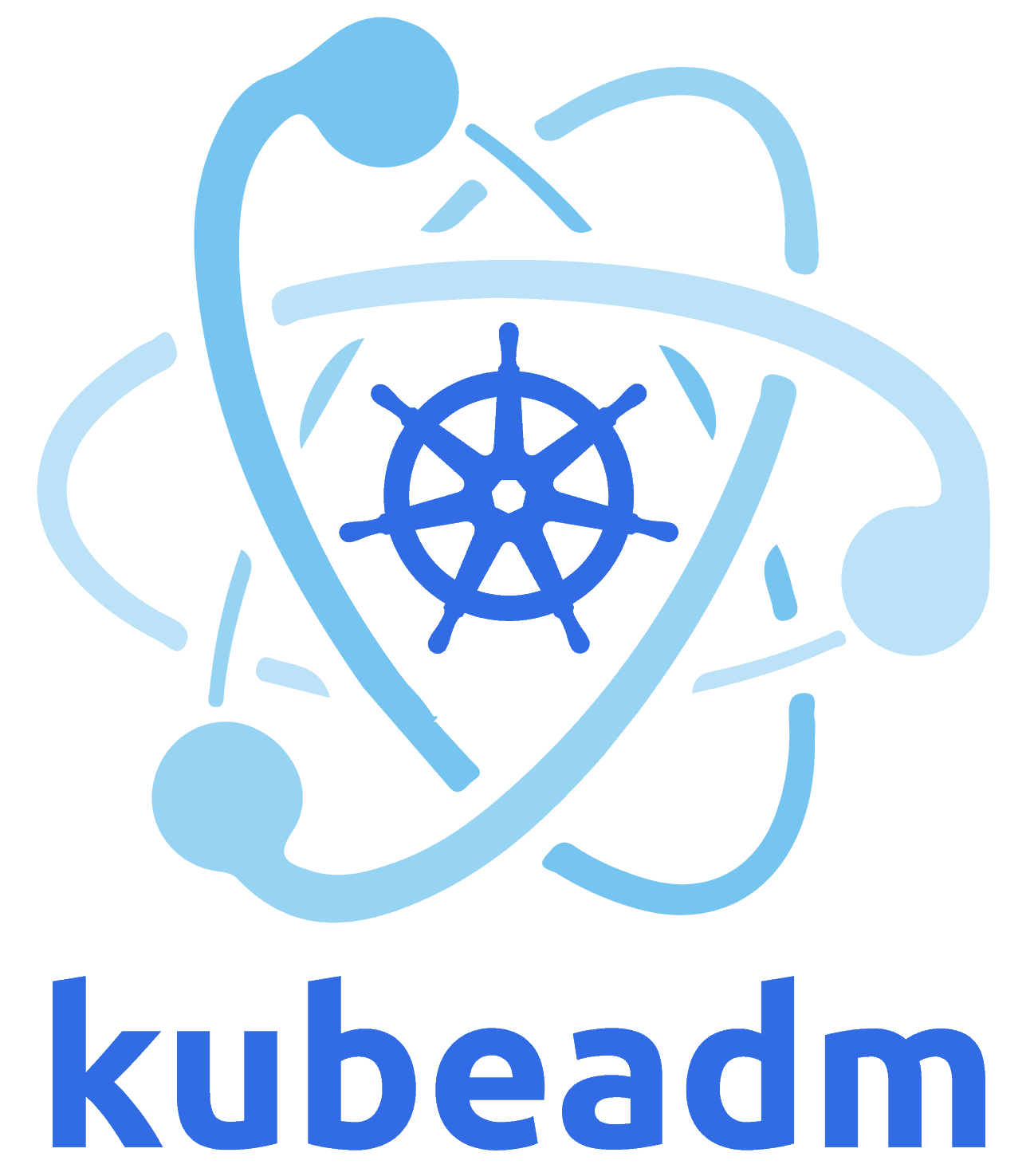
In unmanaged Kubernetes installation, everything must be managed by ourselves, which means that both the master node and worker node are managed by us. It is not managed by a cloud vendor hence, known as unmanaged or installer based.

1. kubeadm
2. Kops
3. Kubespray

**Check this out**: Since Kubernetes is the most ‘hot & trending in the containerization world, most of IT professionals are Kubernetes certification aspirants because of the undisputed value it holds. Let’s come to the good news, Certified Kubernetes Administrator (CKA)

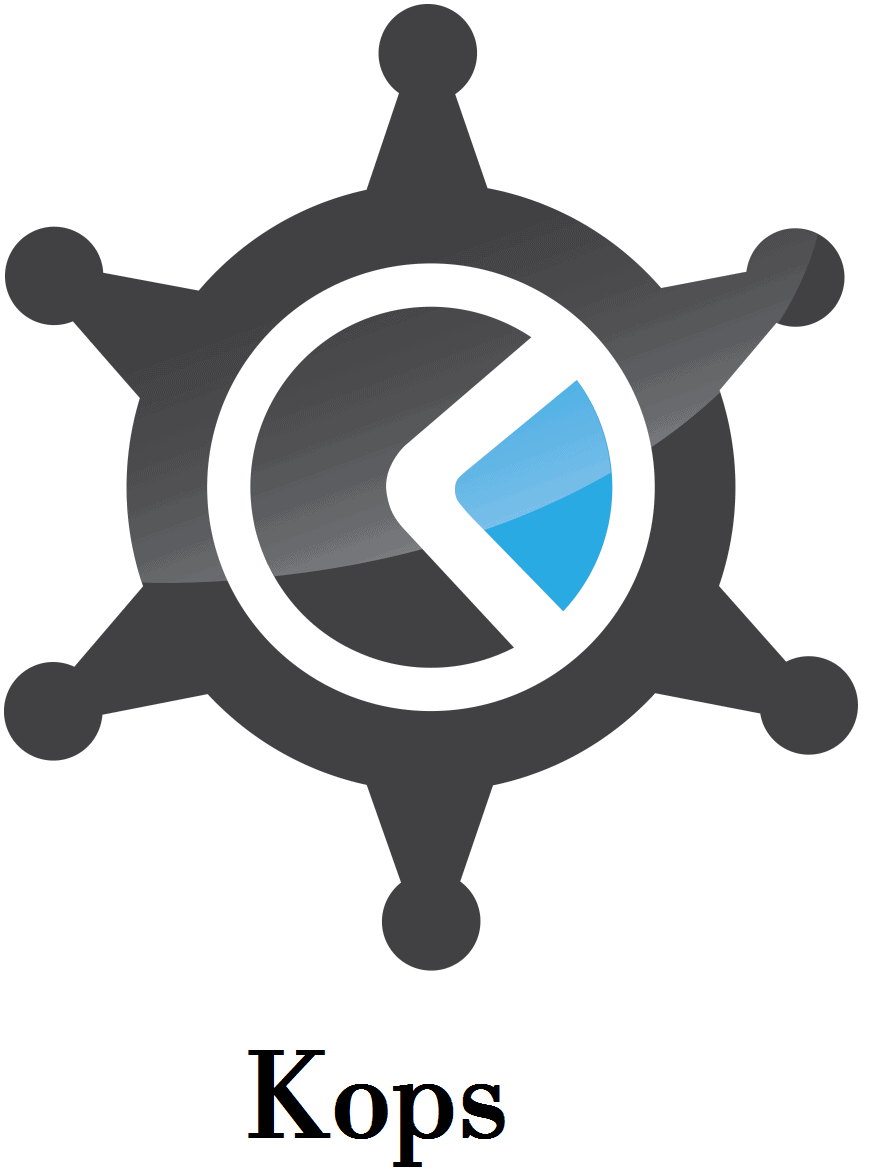
**Create a Kubernetes Cluster using Kubeadm**

It is a tool built to provide **kubeadm init** and kubeadm join as best-practice “fast paths” for creating Kubernetes clusters. It performs the actions necessary to get a**minimum** viable cluster up and running. By design, it cares only about bootstrapping, not about provisioning machines. We can use kubeadm for creating a production-grade Kubernetes environment.

****

**Create a Kubernetes Cluster using Kops**

**Kubernetes** Operations, or **Kops**, is an open-source project used to set up **Kubernetes** clusters easily and swiftly. It’s considered the “**kubectl**” way of creating clusters. **Kops** allows deployment of highly available **Kubernetes** clusters on AWS.

****

**Create a Kubernetes Cluster Using Kubespray**

Kubernetes clusters can be created using various automation tools. Kubespray is a combination of Kubernetes and Ansible. That means we can install Kubernetes using Ansible. We can also deploy clusters using kubespray cloud compute services like EC2 (AWS). Kubespray provides deployment flexibility. It allows you to deploy a cluster quickly and customize all aspects of the implementation*.*

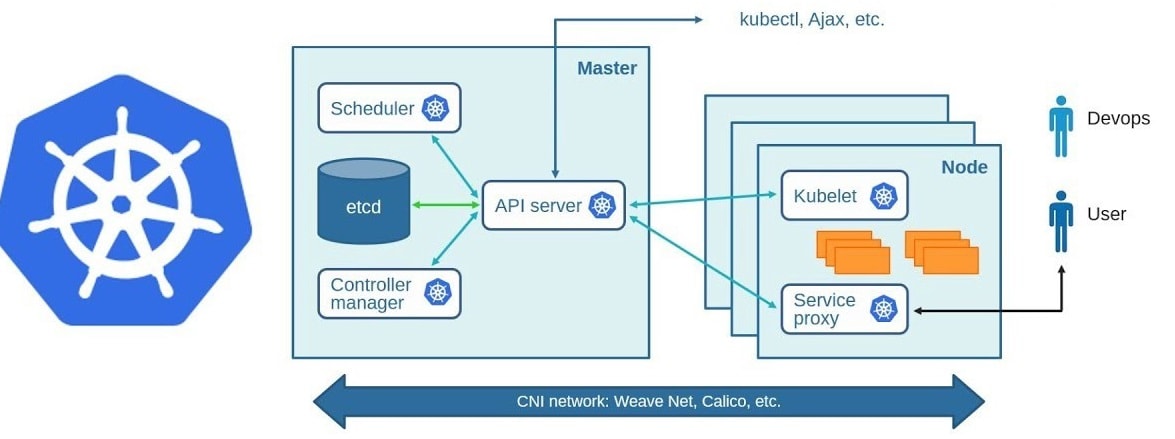
****

**Kubernetes for Development**

This is used for the single node or a quick setup for Kubernetes. For the development purpose, we put everything on a single node. This is limited to one node.

1. Kubernetes using Minikube
2. Kubernetes using Docker for Desktop

**Kubernetes using Minikube (Single Host)**

It is the name of a **go program** and is a lightweight Kubernetes implementation that builds a **Kubernetes cluster** in a single host with a set of small resources to run a small Kubernetes deployment. It is meant for testing scenarios of**Kubernetes** (creating pods, services, managing storage, network ingress rules, etc) but in the local environment for the developer or administrator to test. It’s not meant for production use, since it runs a virtual box, installs Docker, and then deploys the essential Kubernetes containers.  It is used to learn Kubernetes. Production systems should use Kubernetes clusters with 3 master nodes to achieve high availability.

Minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes. All you need is Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away: minikube start.

**Kubernetes Using Docker for Desktop**

Kubernetes with Docker Desktop is for a single node. It is available for Windows, Mac. We use this locally on our systems. We do it as a developer sandbox. This is convenient and easy to install and is primarily used for testing purposes. But docker desktop can’t be used for the production.

**Kubernetes in Production**

When developers begin to experiment with Kubernetes, they would like to deploy Kubernetes on a set of servers. A production-ready Kubernetes environment is one that is ready to **start serving** traffic so for this, we need more than a **single node** cluster. Hence, at a minimum, we would need **one master** node and **2 worker** nodes.  These are the ways Kubernetes is used in production.

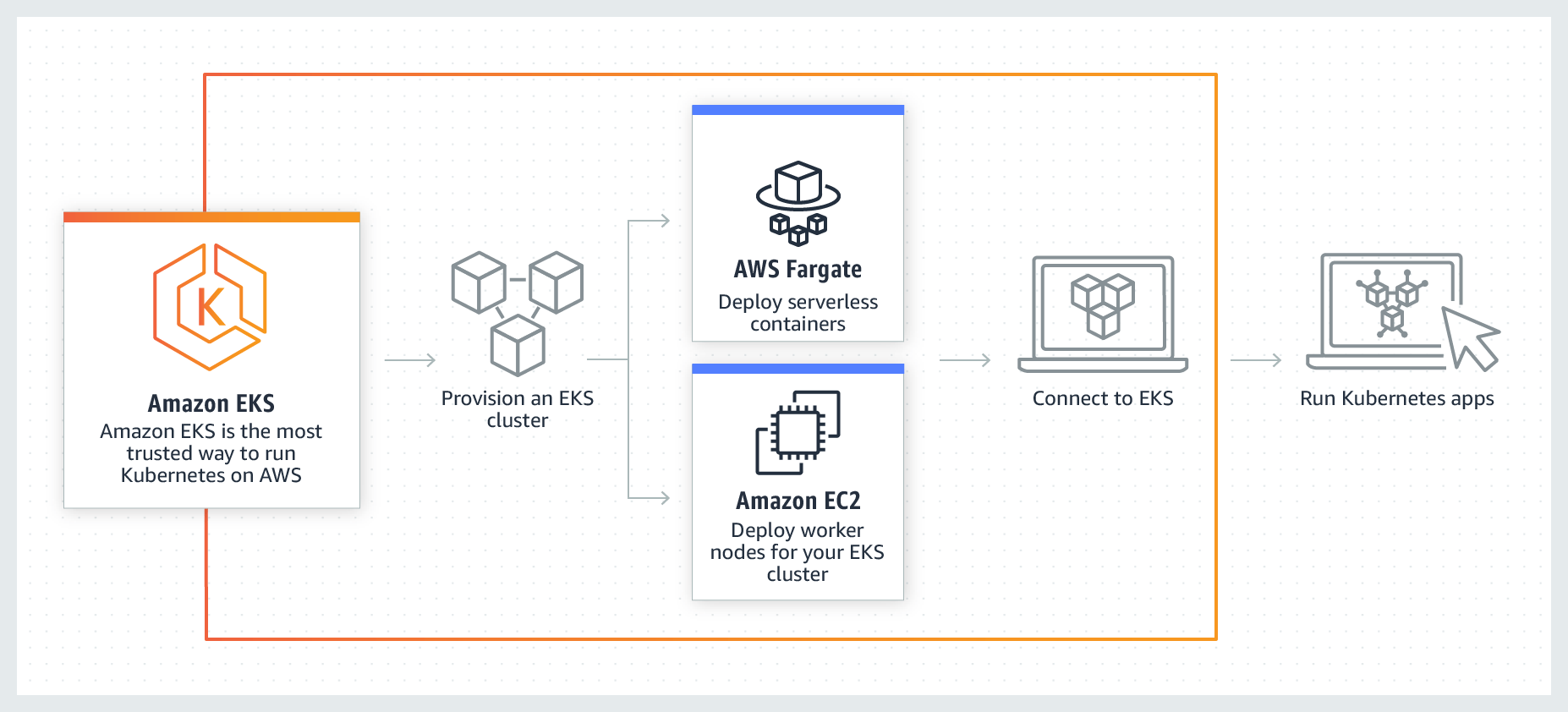
* Managed **Kubernetes** Service Ensures SLA and Simplifies Operations.
* It is used in Cluster Monitoring and Logging.
* Kubernetes is used as Registry and Package Management — Helm/Terraform.
* This is used in CI/CD Toolchain for DevOps.
* Cluster Provisioning and Load Balancing
* It is used for security purposes.
* This is used in Governance.

**Managed Kubernetes (EKS, AKS, GKE, OKE)**

In Managed Kubernetes the nodes are managed by the cloud vendor or managed Kubernetes platform or the platform which basically gives us the Kubernetes configuration It is when **third-party** providers take over responsibility for some or all the work necessary for the successful set-up and operation of K8s. The popular 3rd party providers are

1. Elastic Kubernetes Service**(EKS)**
2. Azure Kubernetes Service**(**[**AKS**](https://k21academy.com/docker-kubernetes/create-aks-cluster-step-by-step-procedure/)**)**
3. Google Kubernetes Engine **(GKE)**
4. Oracle Kubernetes Engine**(OKE)**

**Amazon Elastic Kubernetes Service (EKS)**



EKS runs Kubernetes on multiple AWS availability zones for high-availability, and AWS manages complete infrastructure.

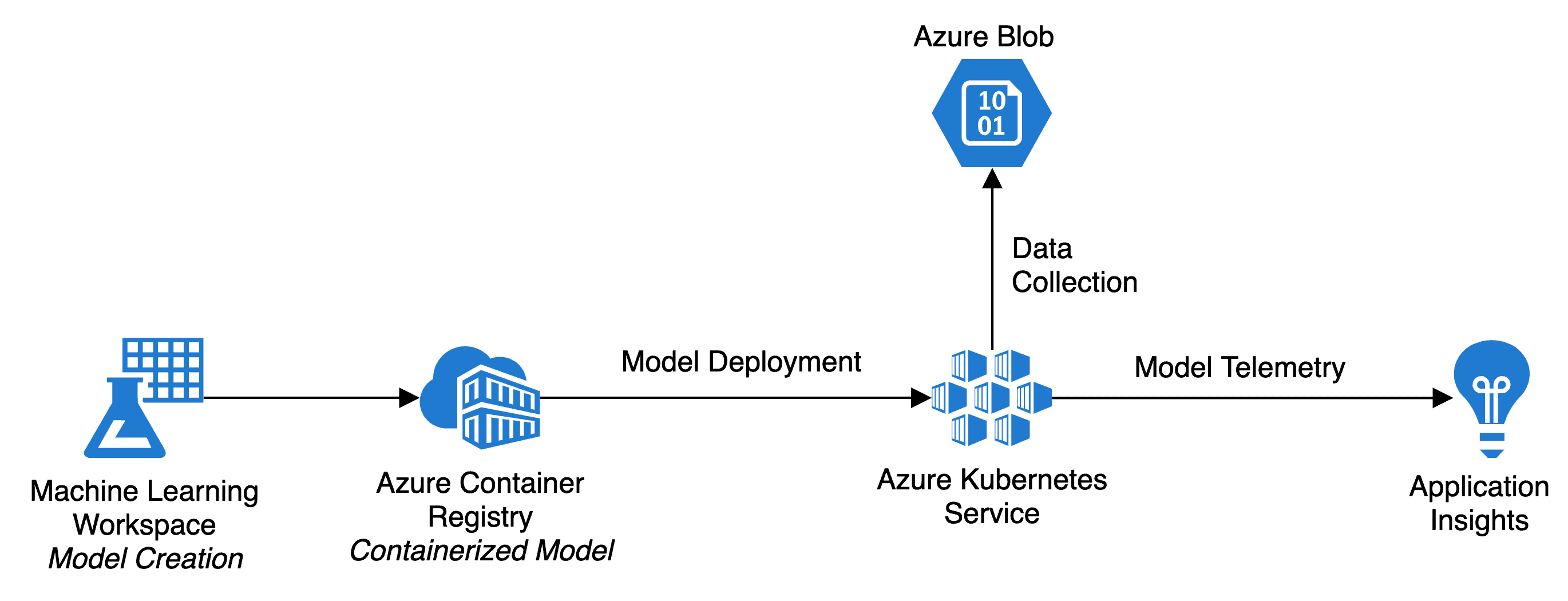
EKS is the best place to run Kubernetes for several reasons. First, you can choose to run your **EKS clusters** using **AWS**

**Fargate**, which is a serverless compute for containers. EKS automatically applies the latest security patches to your cluster **control plane**.

Some of the great EKS features are:

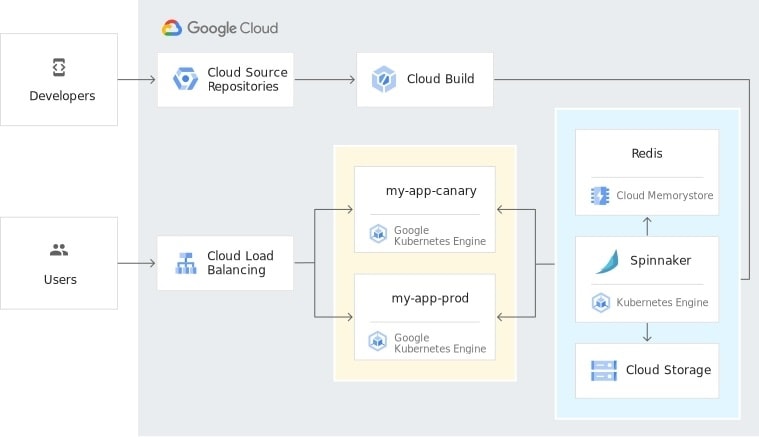
* Manage through web UI or CLI.
* Optimized AMI with NVIDIA drivers for advanced computational power.
* Run a cluster behind AWS load balancer.

**Azure Kubernetes Service (AKS)**



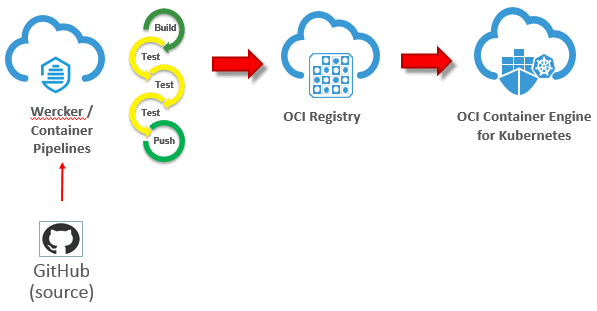
Azure offers **multiple** ways to provision a cluster – web console, command line, Azure resource manager, Terraform. You can take **advantage** of Azure traffic manager to route the application requests to the nearest data centers for a fast response. Deploy and manage containerized applications more easily with a fully managed Kubernetes service. Unite your development and operations teams on a single platform to rapidly build, deliver, and scale applications with confidence. One down-side of EKS is that it is currently unable to support hybrid cloud configurations.

**Google Kubernetes Engine (GKE)**



Since K8s was created by **Google engineers** for in-house container orchestration, it makes sense that GKE is one of the most advanced managed platforms available. Designed for use on Google Cloud, it includes functionality for operation in hybrid environments as well. It allows you to transfer microservices with minimal configuration changes, create private image repositories via an integrated image builder, and manage authentication and access rights through an integrated console.

**Oracle Kubernetes Engine (OKE)**



**Oracle Cloud** Infrastructure Container Engine for Kubernetes is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications to the cloud. Use Container Engine for Kubernetes (OKE) when your development team wants to reliably build, deploy, and manage cloud-native applications.

**Kubernetes Operator: An Overview, Stateful Application Example**

Do you know why is Kubernetes is the talk of the hour? We need to look at the advent of containers and microservices, which brought us excellent software development and infrastructure capabilities. K8s performs automation with ease, and the Kubernetes Operator aids them a lot.

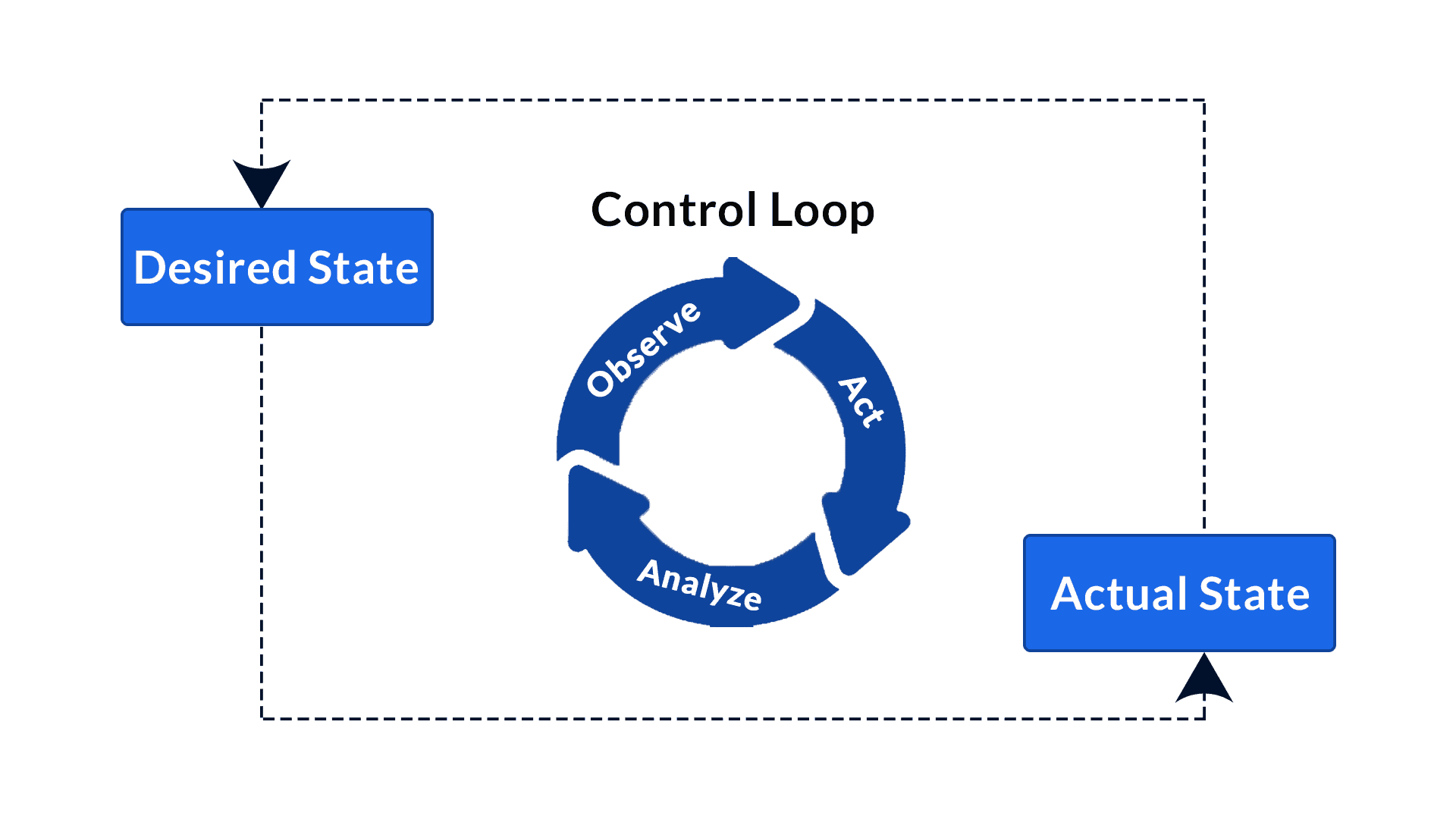
Kubernetes’s idea was to automate stuff, but then if we need Operators, what’s the whole point of K8s? This blog helps you decode the need for and importance of Operators. These are the following topics that I will be addressing in this post:

* [Control Loop](https://k21academy.com/docker-kubernetes/kubernetes-operator/#Stateful_Application_without_Operator)
* [What is an Operator?](https://k21academy.com/docker-kubernetes/kubernetes-operator/#The_concept_of_the_desired_state_comes_from_tracking_a_Kubernetes_resource_type_The_desired_state_of_an_object_is_present_in_the_spec_field_So_the_controller_s_responsible_for_bringing_the_cluster_from_the_observed_state_to_the_desired_state_A_controller_may_decide_to_perform_this_action_independently_But_usually_the_controller_in_K8s_will_communicate_with_the_API_server_to_take_the_necessary_measures)
* [How does a Kubernetes Operator work?](https://k21academy.com/docker-kubernetes/kubernetes-operator/#TL_DR)
* [Why Use Operator?](https://k21academy.com/docker-kubernetes/kubernetes-operator/#Everything_in_Kubernetes_runs_as_containers_Operators_are_no_exception_Below_is_an_example_of_an_Ambassador_pattern_multi_container_and_an_operator_An_operator_consists_of_a_code_that_performs_the_commands_It_also_has_a_Custom_Resource_Definition_CRD_which_maps_the_operator_code_back_to_the_kubectl_command)
* [When to Use Operator?](https://k21academy.com/docker-kubernetes/kubernetes-operator/#Our_ideas_and_knowledge_can_be_defined_in_an_operator)
* [Stateless Application on Kubernetes](https://k21academy.com/docker-kubernetes/kubernetes-operator/#When_you_need_to_encapsulate_a_stateful_application_business_logic_controlling_everything_with_Kubernetes_API)
* [Stateful Application without Operator](https://k21academy.com/docker-kubernetes/kubernetes-operator/#Therefore_you_don_t_need_to_sit_and_control_the_application_once_deployed_nor_need_a_controller_Kubernetes_handles_all_of_these_tasks_because_it_is_a_part_of_the_control_loop_mechanism_It_observes_the_current_state_and_knows_the_desired_state_by_the_deployment_file_it_updates_the_configuration_to_the_desired_state)
* [Stateful Application with Operator](https://k21academy.com/docker-kubernetes/kubernetes-operator/#If_it_is_the_case_that_we_always_need_manual_intervention_doesn_t_matter_if_they_run_on_K8s_or_traditional_servers_So_we_require_people_who_operate_these_applications)

Before we hop on to Operators, let’s first understand what a Control Loop is.

**Control Loop**

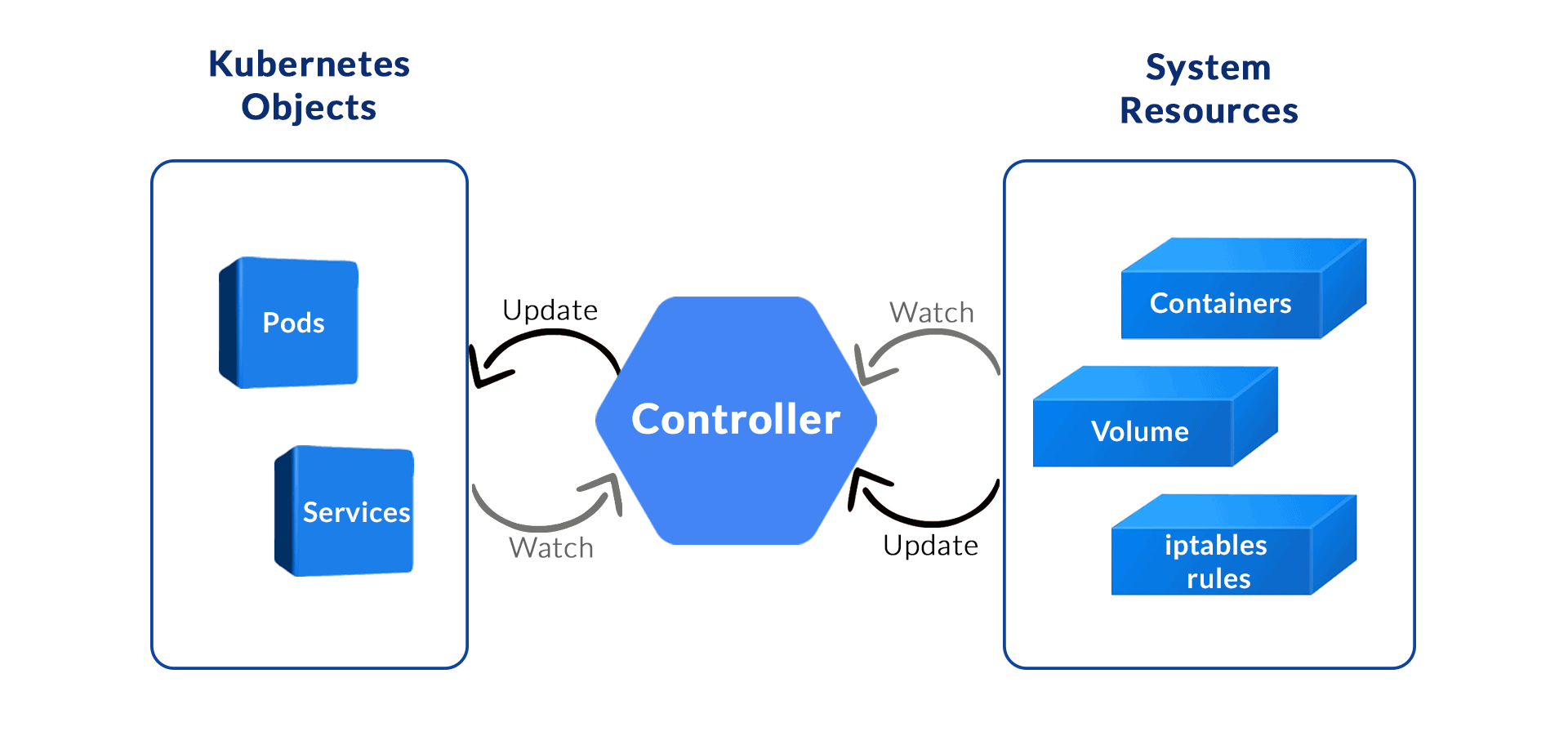
Controllers are no exception in the field of Kubernetes. Since it is all about automation, there has to be an inevitable controller listening to which the K8s runs accordingly. Hence, the controllers reduce the sysadmin toil.



Control Loop

In short, controllers are the control loops that observe the state of a K8s cluster. As a result, make or request changes where required. Every controller strives to move the current cluster state nearer to the desired state.

**Controller Pattern**

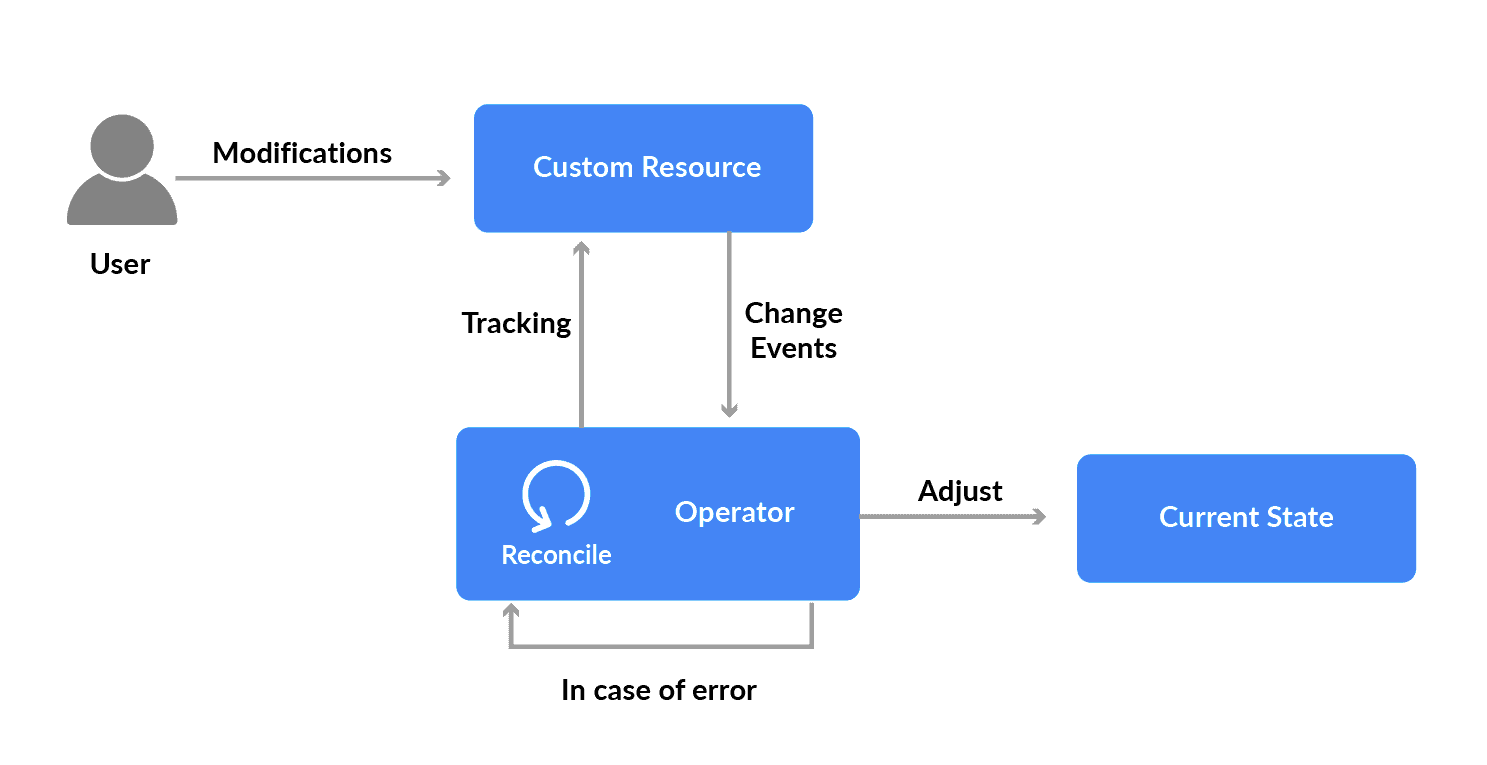


Controller

The concept of the desired state comes from tracking a Kubernetes resource type. The desired state of an object is present in the spec field. So, the controller/s responsible for bringing the cluster from the observed state to the desired state. A controller may decide to perform this action independently. But usually, the controller in K8s will communicate with the API server to take the necessary measures.

**What is a Kubernetes Operator?**

The best practice says that automation is not interfered with by Humans as they are prone to errors. Also, if it comes to us to fix a broken pod in a cluster, scale up and down applications, it defeats the whole point of container automation. For ‘package, it since and run it everywhere to be accurate, there are many other things to address.

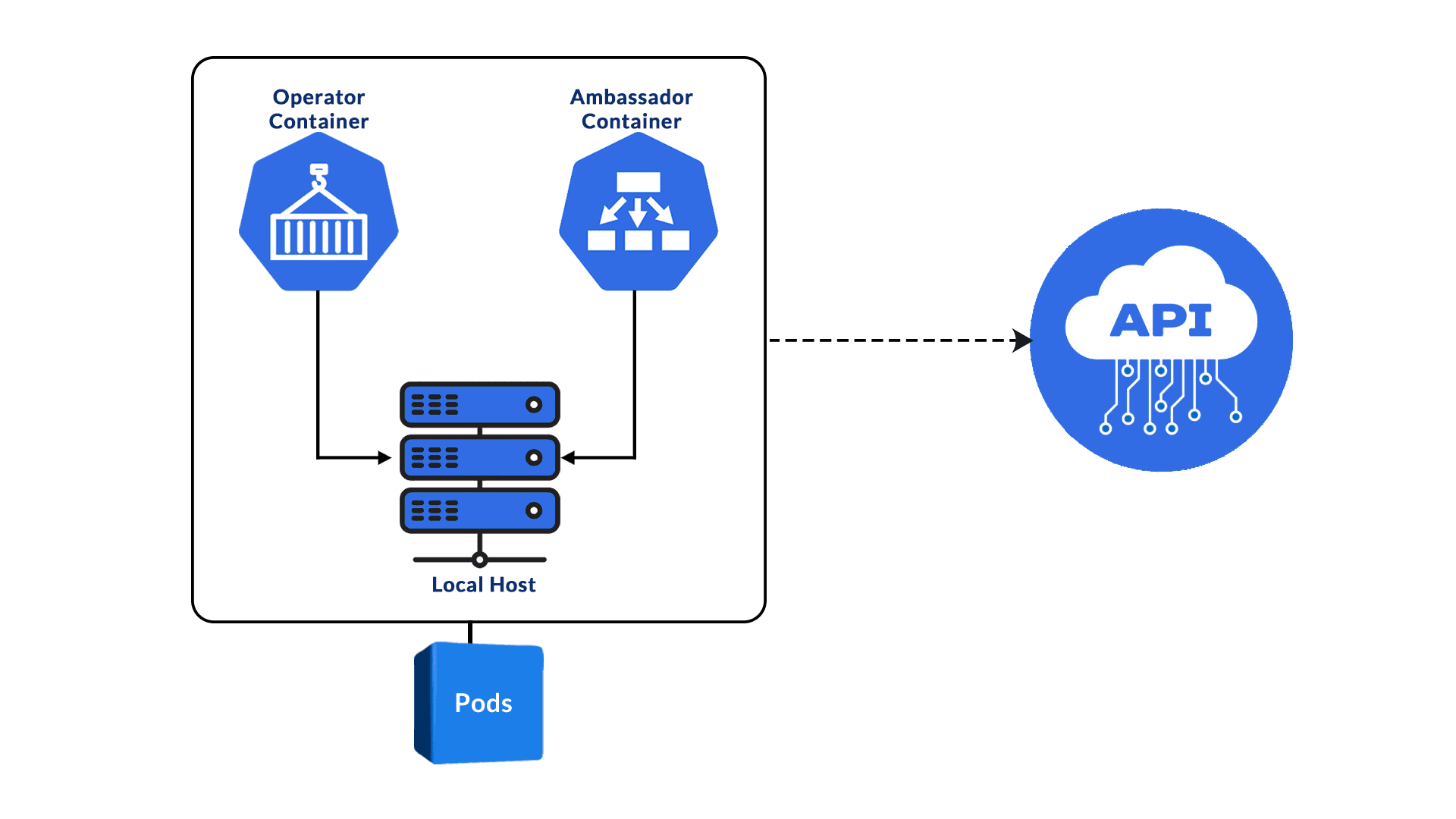


Kubernetes Operator

An operator in Kubernetes is an application-specific controller. Kubernetes Operators are application-specific and are software written to encapsulate all the operational considerations of an application. It extends a Kubernetes API to create, configure, and manage complicated applications in place of humans. Let us look at an example where an operator deploys a database, upgrades database versions, or performs backups. Therefore, these systems can then be tested and be relied on to react quicker than a human engineer could.

**How does a Kubernetes Operator work?**

The operator is nothing but software that does everything a human operator does. So all the tasks of a DevOps member/team are packed into the software. Some of the instructions of deploying an application, creating a cluster of replicas, recovery instructions, etc. It always keeps a watch and gets the application to the desired state.



Kubernetes Operator Architecture

At its core, the Kubernetes operator has the exact control loop mechanism. It handles the recovering of an application or restarting an application because of the update in deployment. So, Kubernetes Operators are responsible for all the actions performed after changes in the environment.

Everything in Kubernetes runs as containers; Operators are no exception. Below is an example of an ambassador pattern multi-container and an operator. An operator consists of a code that performs the commands. It also has a Custom Resource Definition (CRD), which maps the operator code back to the kubectl command.

* The operator container contains the program that sees the API and identifies the changes.
* The Ambassador container runs kubectl proxy. It helps in connecting the operator container with the K8s API server.

**Why use Kubernetes Operator?**

Let’s look at some reasons why the Kubernetes operators are needed:

* The Operators extend Kubernetes functionality.
* Our ideas and knowledge can be defined in an operator.
* K8s Operators do this in a scalable, repeatable, standardised fashion.
* Operators improve resiliency while reducing the burden on IT teams.
* Operators prove particularly useful in multi-cloud and hybrid cloud environments.

**When to use a Kubernetes Operator?**

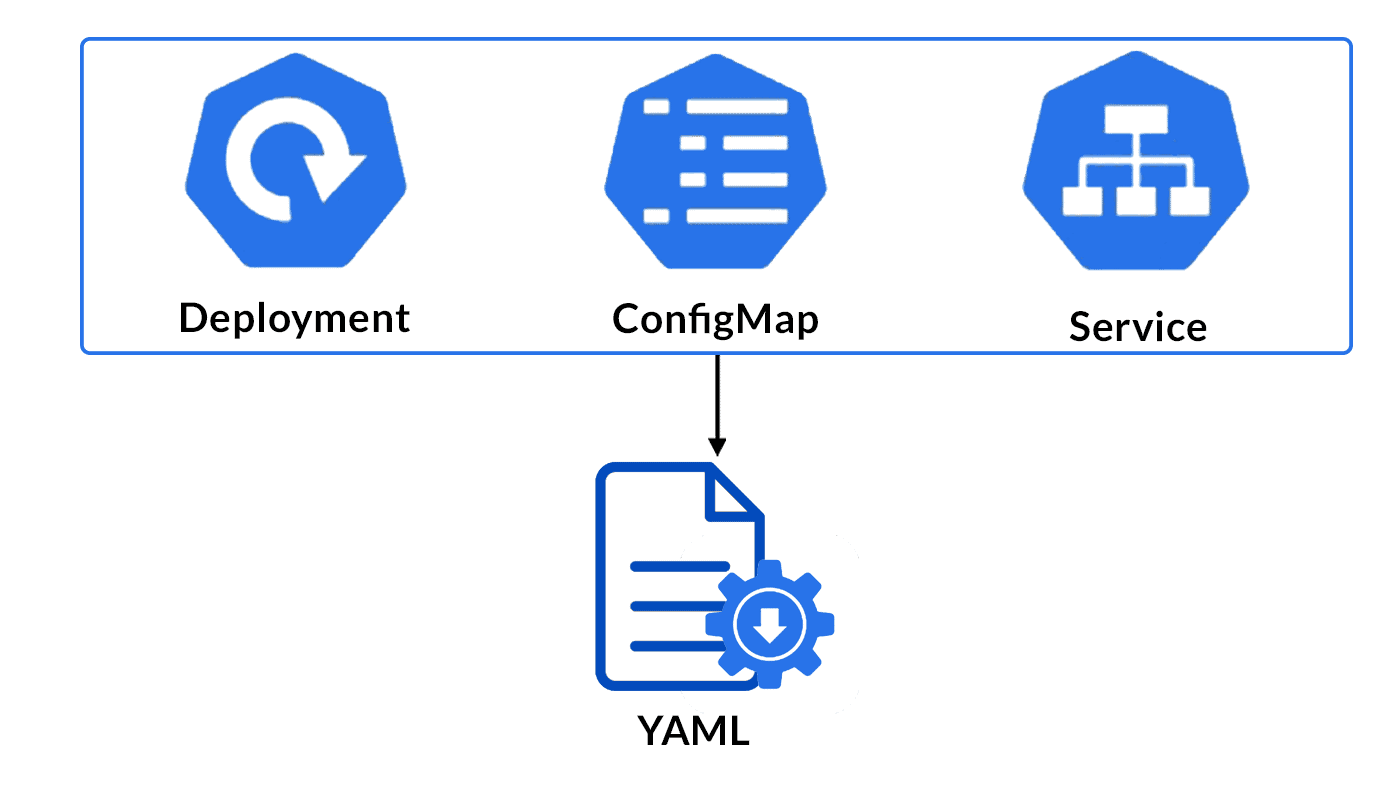
FYI, ‘*All operators are controllers, but not all controllers are operators* ‘. We can choose an existing Operator; on the other hand, you can write it too. But writing operator code will also constrain you from maintaining it. Also, it means that you need more developers to keep the environment robust and secure.

It would be best if you used an operator under the following conditions:

* When you need to encapsulate a stateful application business logic controlling everything with Kubernetes API.
* When you need to build a tool that watches your applications for changes and performs specific SRE/Ops tasks at required times.

Now let’s look at some real-time scenarios where we use Kubernetes Operators that will come into deployment.

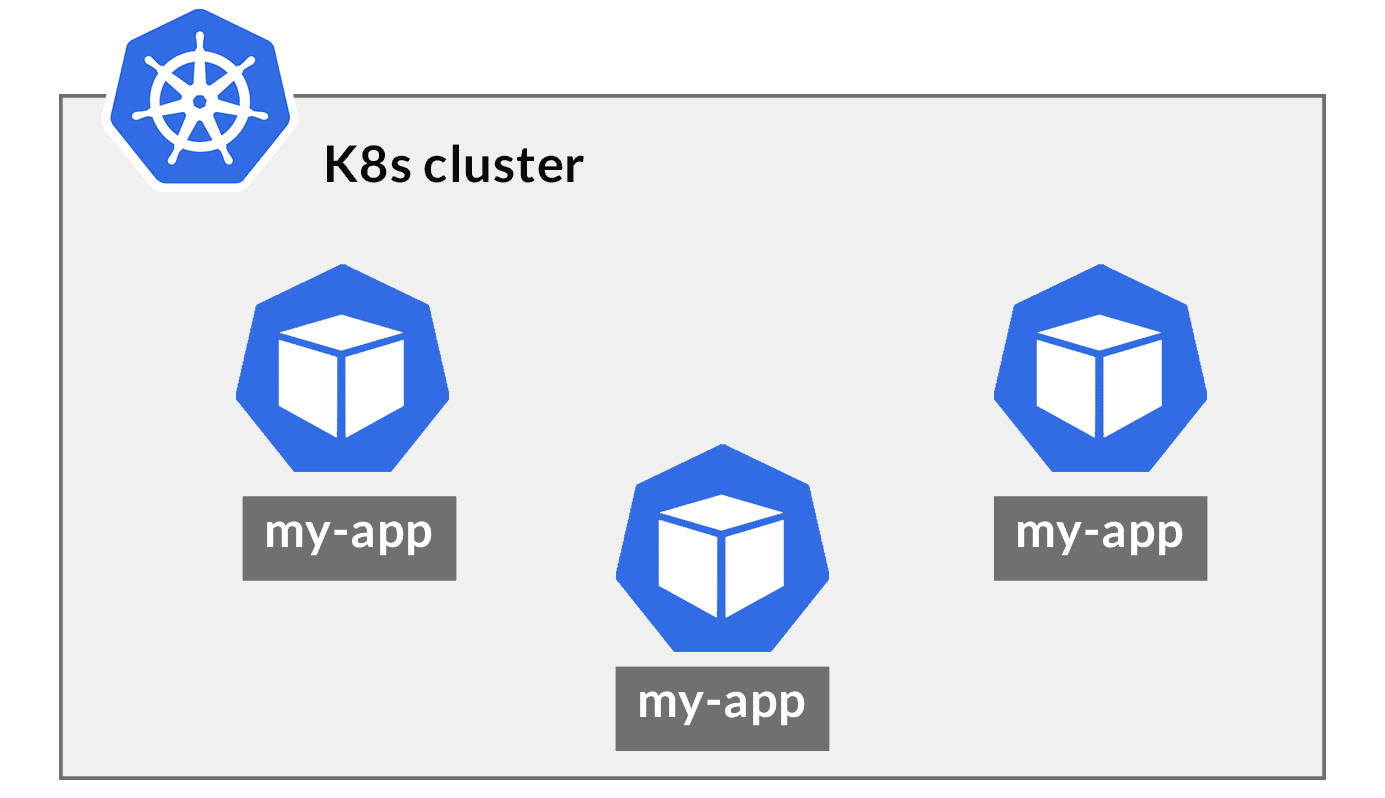
**Stateless Application on Kubernetes**



Application Configuration

The first example use case is a web application. You write your deployment, configmap. And service files as per the requirement and spin up the applications. You also scale the application to the preferred number. In our case, we are scaling it to 3 replicas.

Let’s say one of your application dies for various reasons. So, this is where the Kubernetes comes in and recovers it using the control loop mechanism we saw earlier. Hence a new application gets created and replaces the crashed application. If you make any changes in the deployment configuration, all replicas restart with the latest version.



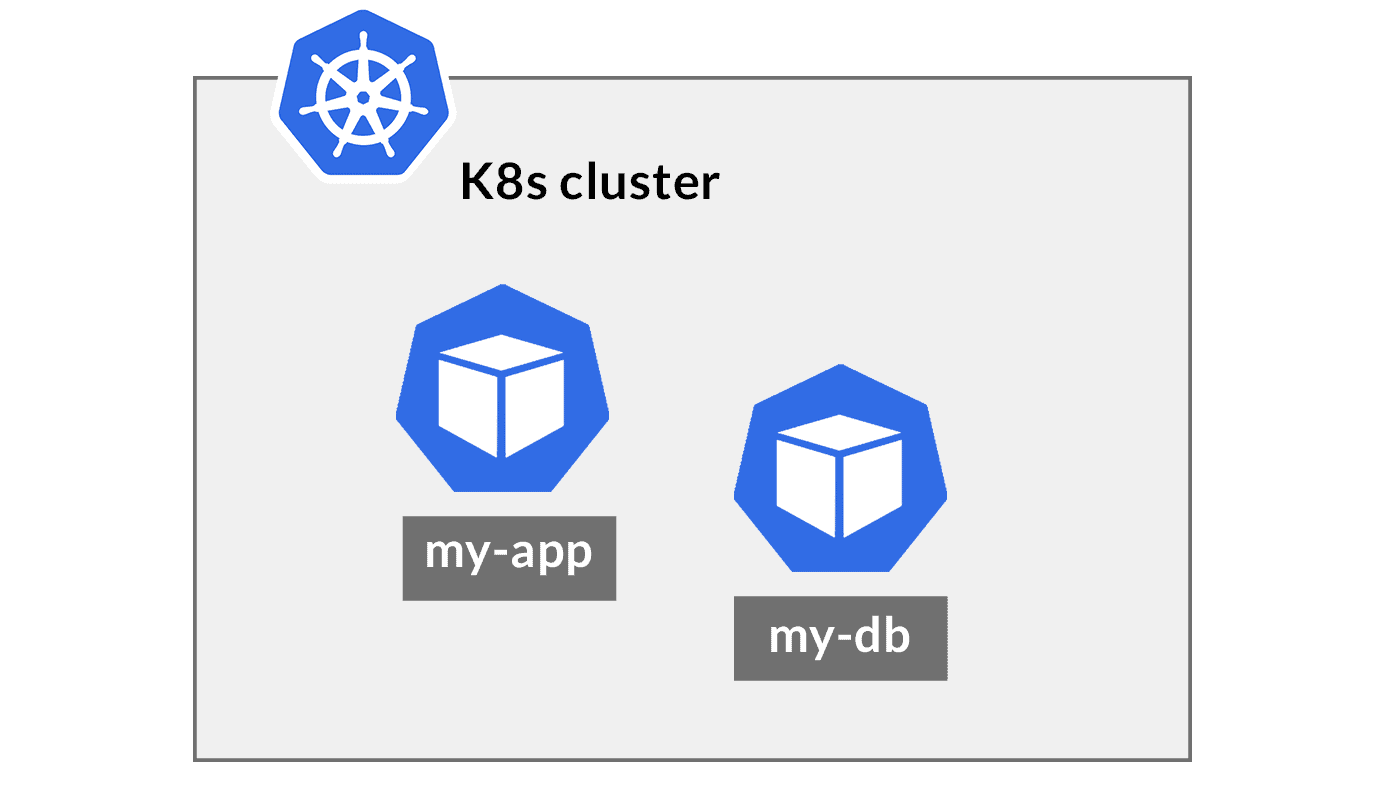
Stateless K8s Application

Should we not take backups? Well, the answer is no, as the application is **Stateless**. So, here is a perfect example of Deploy once; run it many times. Hence, when you update the deployment or Scale-Up/Down the application, it works pretty much the same without any problems.

Therefore, you don’t need to sit and control the application once deployed, nor need a controller! Kubernetes handles all these tasks because it is a part of the control loop mechanism. It observes the current state and knows the desired state by the deployment file; it updates the configuration to the desired state.

**Stateful Application without Kubernetes Operator**

Now, we have a web application that requires a database of data persistence. The database applications always possess some state, and we must keep a watch throughout the lifecycle. These applications are called the **Stateful** applications.



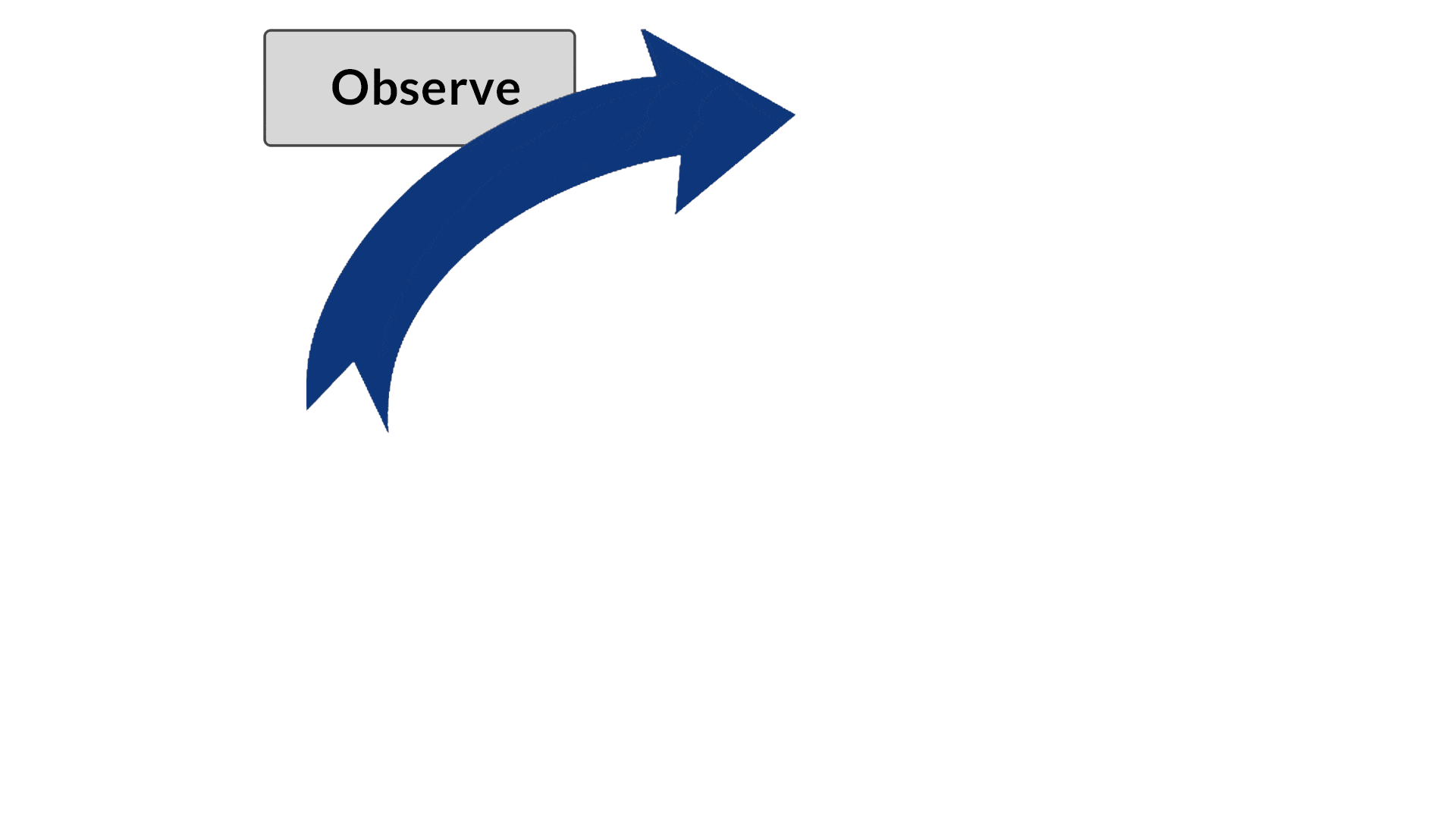
Stateful K8s Application

So, if you decide to create three replicas of a database application, their states and identities will differ. Hence, it is a challenge for Kubernetes to handle everything because all the replicas should communicate with each other and be in sync. Also, different databases have different workarounds, so we can’t have a generalised solution for this.

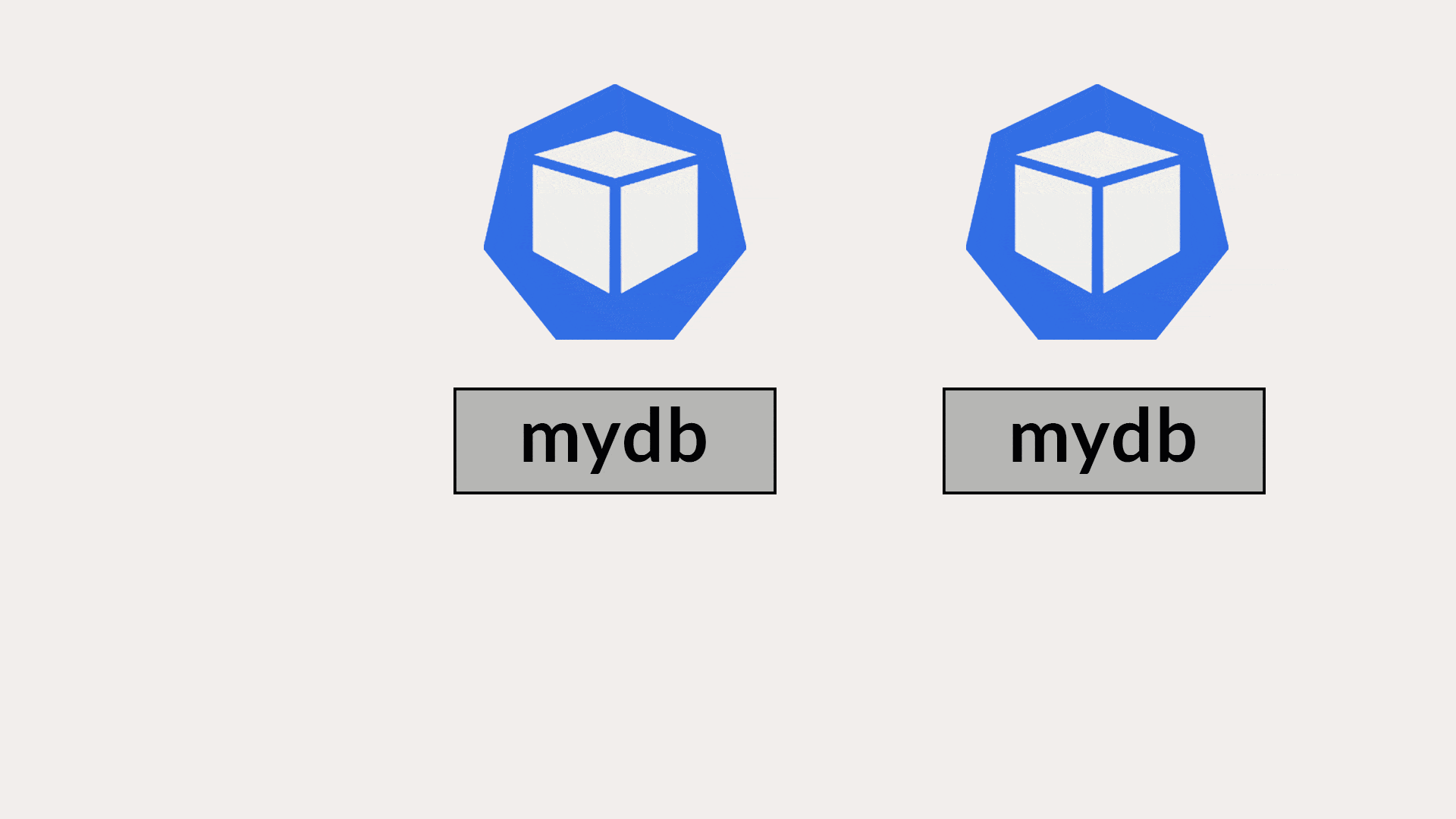
If it is the case that we always need manual intervention doesn’t matter if they run on K8s or traditional servers. So, we require people who “**operate”**these applications.

But the manual intervention of K8s voids the idea of Kubernetes! The concepts of automation, load-balancing, self-healing capabilities, etc. Don’t jump the gun so quickly; the next section is all the reason you have come here!

**Stateful Application with Kubernetes Operator**



So, the time has finally come to see an Operator in action. We shall see how the operator enhances the same web application. Using an application-specific operator makes the job easier and effortless after deployment. We don’t need a manual intervention unless there is a significant change in the whole system. Operators are reusable and have a single automated process for all the available similar Kubernetes cluster.



Stateful Application with Kubernetes Operator

The more challenging the application deployment gets, the harder it becomes for a human operator. Hence, a human operator vs a Kubernetes operator is a no brainer.

So, if an application dies, the operator recovers it without any downtime as it always keeps a watch of the current state. Also, any change in the deployment, the Kubernetes Operator is responsible for restarting the latest versions.