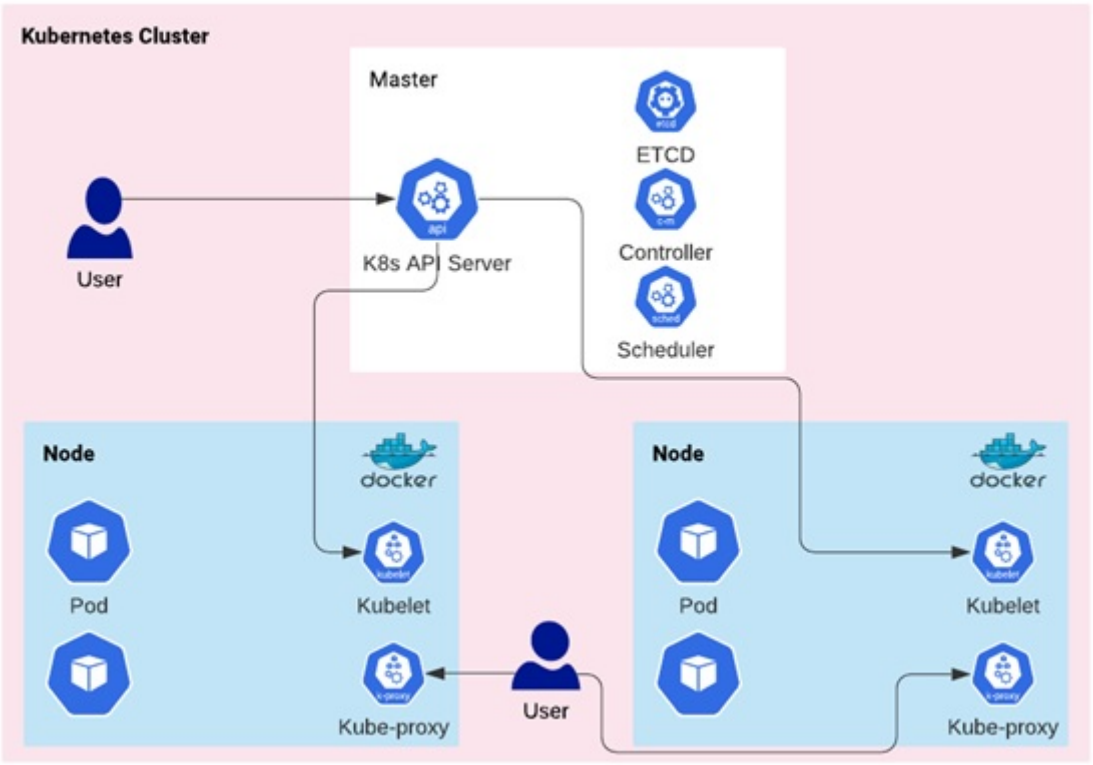
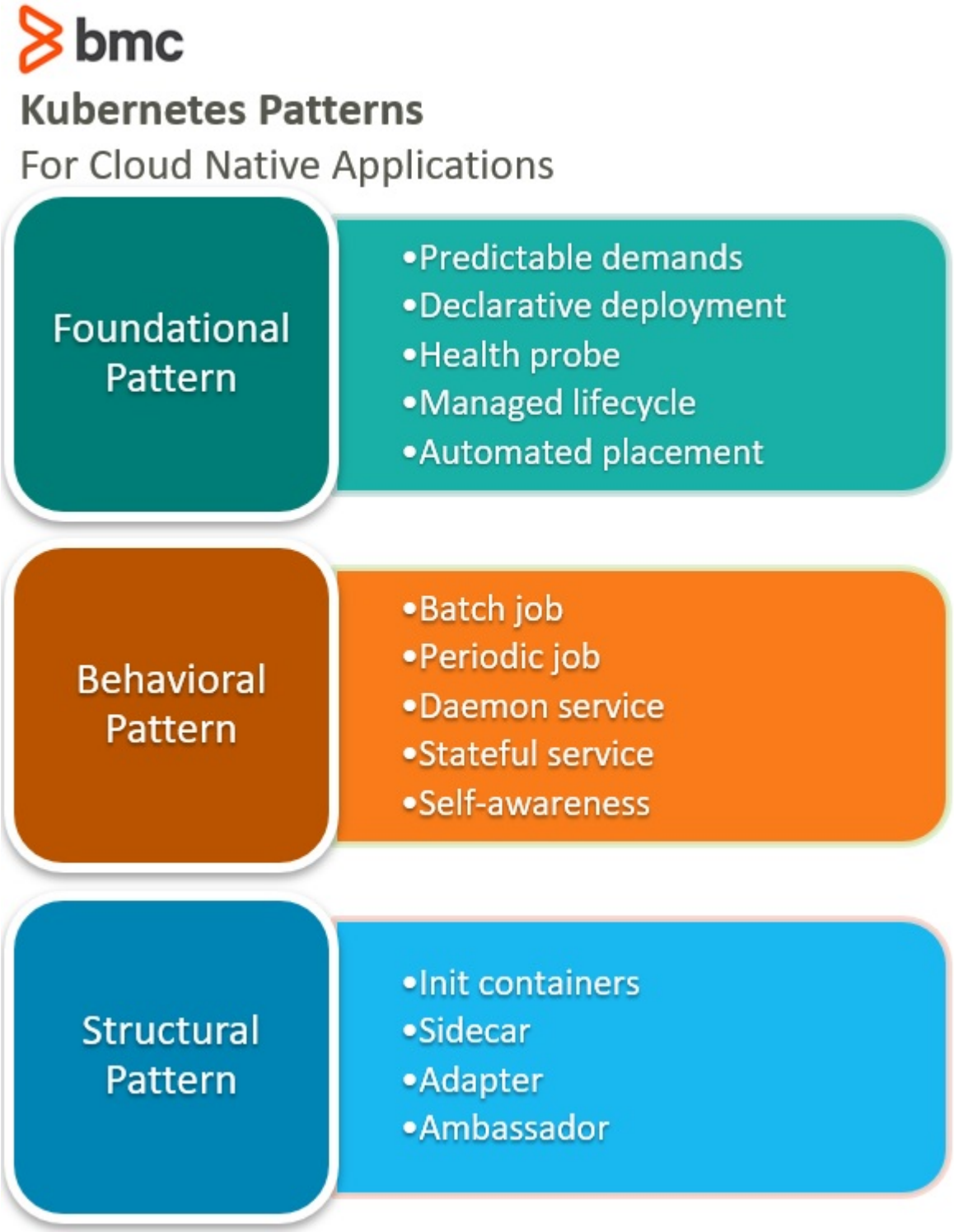
3 KUBERNETES PATTERNS FOR CLOUD NATIVE APPLICATIONS

Modern cloud native applications are designed to be scalable, elastic, reliable, and predictable. To achieve these characteristics, certain principles can help guide admins or architects when running applications in Kubernetes.



(Source)

In this article, we are going to cover three important patterns to consider when running a cloud native application on Kubernetes. At the end of this post you should have a good understanding of what to look for when architecting a containerized application that is going to run inside of a K8s cluster.



(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

Foundational Pattern

The foundational pattern is fundamental to running any container-based application in a Kubernetes cluster and for it to be considered cloud native.

There are five areas to focus on when working with the foundational pattern:

- Predictable demand
- Declarative deployment
- Health probe
- Managed lifecycle
- Automated placement

These concepts cover most of what you would need to understand to run your application in Kubernetes with this pattern.

Predictable demands

Understanding applications resources and runtime dependency is key to running any application on Kubernetes. To be able to predict the demand of the application we need to know:

- The runtime dependency (e.g., persistent volume or ConfigMap)
- The application resource profile which covers setting compressible resources (CPU) and incompressible resources (memory)

Declarative deployment

Automating application deployment is another critical part of running a cloud native app on Kubernetes. This principle is only possible with the use of the deployment workload to provide declarative updates for pods and replicasets. Similarly, for a deployment to do its job correctly, it expects the containerized applications to be good cloud-native citizens.

At the very core of a deployment is the ability to start and stop a set of pods predictably. For this to work as expected, the containers themselves usually listen and honor lifecycle events such as sigterm and also provide health-check endpoints which indicate whether they started successfully.

Health probe

A cloud native Kubernetes application must be observable. Health probe concept focuses more on how the running application should communicate its state to the cluster. Using a combination of process health check, liveness probe, and readiness probe, you can quickly determine if application is healthy and how to control traffic.

In the case where the liveness probe fails, the container is completely terminated but with the readiness probe failure, the container is removed from the service.

Managed lifecycle

Container based applications running in k8s should be able to react to certain lifecycle events emitted by the cluster. Understanding how to take advantage of process signals like sigterm and sigkill (force kill after 30 seconds), poststart, and prestop hooks is important.

- **Poststart hook** is a blocking call that executes when a container gets created. Pod status is pending until poststart hook is completed and if poststart hook fails, the container process gets terminated.
- **Prestop** is a blocking call sent to a container before termination and it is particularly useful for shutting down a container gracefully based on events emitted by the cluster.

Automated placement

Understanding how container-based applications will be distributed on a Kubernetes node is key to performance, reliability, and automation. The job of a scheduler is to retrieve each newly created pod definition from the API server and assign it to a node. However, it is not as simple as that—we must make sure that node capacity will allow pod placements based on container resource profile.

Things to consider when dealing with automated placements are:

- Available node resources
- Container resource demands
- Placement policies
- Node affinity, pod affinity and anti-affinity
- Taints and tolerance

Behavioral Pattern

The concepts in the behavioral pattern are focused around the pod management primitives and their behaviors. It is very important to know how you want to run your application in a pod in a cluster depending on the use case.

For example, you can run an application in a bare pod or as a daemonset which runs a single pod in every node in the cluster.

The concepts covered under the behavioral pattern are:

- Batch job
- Periodic job
- Daemon service
- Stateful service
- Self-awareness

Batch job

This is used to run short-lived pods reliably until completion. This is particularly useful when you don't want a long running pod, but instead want to simply run a task that runs to completion and terminate the pod. Unlike other pod primitives, batch jobs will not restart pods once they run to completion.

Periodic job

This is used to execute a unit of work to be triggered by an event or at a certain time. Periodic jobs are deployed using the cronjob primitive where you can think of cronjobs as one line of a unix crontab. The difference between Kubernetes cronjob and Unix cron is that Kubernetes cronjob is highly available and allows both time and event based execution. This approach is useful when applications need to run a specific task like file transfer or polling another service.

Daemon service

In general daemons are programs that run as processes without interaction. Since Kubernetes is a

distributed system and applications running on it are also distributed by nature, daemonset can be used to provide insight into the cluster or application running on it, e.g., running fluentd daemonset on every node to collect cluster logs.

Stateful service

This concept is for applications that are not stateless. Unlike a typical 12 factor application where application can be terminated and expected to be restarted with any issue, a stateful application is the opposite. It is mostly for applications that need to store state. By using statefulset workload, you can control persistent identity, networking, storage, and ordinality, which impacts how application is scaled up and down.

Self-awareness

In some cases, applications running in Kubernetes need to know certain information about themselves, this is what the self-awareness concept solves with downward API. It provides a simple way for introspection and metadata injection into applications running in Kubernetes. Downward API allows passing metadata about the pod to the containers and the cluster through environment variables (e.g., POD_IP) or files.

The ability for an application to be aware of its own surroundings is powerful when you think about how distributed Kubernetes is by nature.

Structural Pattern

This pattern is one of the most important patterns to understand when running any distributed cloud native application on Kubernetes. The main objective of the structural pattern is to help understand how to organize containers in a pod to solve different use cases.

The core of this pattern is based on a single responsibility model where containers should only perform one task. If it needs to do more than one single task, there should be a second container to perform the second task.

To implement this pattern, there are four areas to focus on depending on the use case:

- Init containers
- Sidecar
- Adapter
- Ambassador

Init containers

Separating initialization related tasks from application startup is the main focus of init containers. For example, the ability to perform database schema setup before starting application.

This concept allows for running two sets of containers in a single pod—the init containers and the application containers. The only difference is that application containers run in parallel while init containers run sequentially. The benefit of this is that you can be assured that the initialization completes successfully before the application starts up.

There is also no readiness check for init containers, which is important to know.

Sidecar

Unlike init containers whose job is only to initialize the application containers during startup, sidecars containers allow us to extend the functionality of an existing container without changing the container. This pattern allows single-purpose containers to cooperate closely together in a distributed system like Kubernetes.

An example of this concept is a pod with two containers where one is an http server and another container pulls down the file the http container is serving, maybe from s3, storage bucket, or git. The behavior of the http container is now enhanced by the other container.

Adapter

This is a variation of the sidecar mentioned above and it is used to solve situations where all containers running in an application pod need to be exposed in a unified way.

For example, we have a pod with three containers where two containers work together as the application and the last container, the adapter, parses and formats the logs of the two application containers in a unified way for a third party tool to consume. The main containers could be written in two languages, but their logs are still presented the same way that other tools can understand.

Ambassador

This is another variation of the sidecar concept. There are times where you need your containerized application running in a pod to communicate with external services outside of the pod. This is where the Ambassador concept comes into play. It acts as a proxy and removes the responsibility of the main application container contacting an external service directly.

For example, a situation where an application has to query an external datastore or cache. In this case a request hits the main application container that sends the request to the ambassador container that forwards the request to the external source.

Useful Kubernetes patterns

The patterns and considerations mentioned above are good patterns to think about when architecting and running distributed cloud native applications in Kubernetes to achieve scalability, elasticity, reliability, and predictability.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program