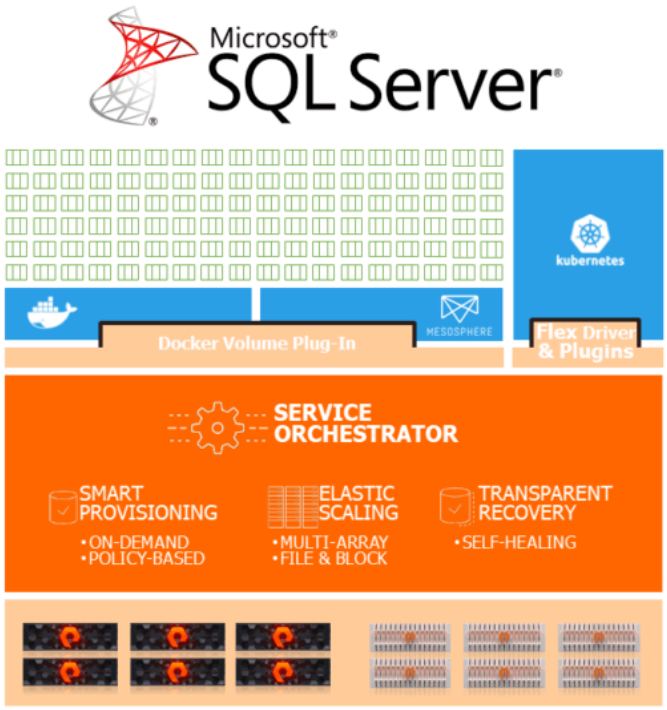
Move on Up Stack With Kubernetes and SQL Server



Containers, popularized by Docker and Kubernetes, born out of Google, are causing a seismic shift in the way in which applications are packaged, deployed and executed.

Why Does This Matter to Microsoft Professionals?

The traction behind containers and Kubernetes has not gone unnoticed by Microsoft:

- The first release of SQL Server 2017 was in Linux container image form,
- The headline new feature in SQL Server 2019; “Big data clusters”, is an analytics platform that runs on Kubernetes,
- Brendan Burns, one of the founding engineers of Kubernetes now works for Microsoft,
- SQL Server availability groups now support Kubernetes.

The messaging from key members of Microsoft’s engineering team to the data platform community is container-and-Kubernetes-centric:

A Key Takeaway Point

Relational databases are perceived as technologies belonging to the “Legacy stack”. Containers and Kubernetes are technologies which belong very much to the “New stack”. The reality is that the legacy and new stacks are converging.

An Overview of Containers

When people first dip their toes in the water with containers, there is a tendency to compare containers to virtual machines. To dispel this notion, Docker’s blog post, “Containers are not VMs”, is recommended reading. The primary concerns addressed by containers are application packaging and delivery, whereas the focus of virtual machines is infrastructure virtualization. The docker container engine runs on most of the popular operating systems and clouds. You can download Docker Community Edition for free. However, there are some crucial things required to run enterprise grade containerized applications that stand alone container engines do not provide:

- Management of secrets and configuration
- Horizontal scaling
- Storage orchestration
- Load balancing and scheduling
- Resilience
- Service discovery

And this is where a container orchestration framework comes into play.

Along Came Kubernetes

Google had a vision of giving developers outside of Google the same development experience that developers inside Google enjoyed. A vision that Joe Beda articulated in an interview at KubeCon 2017. Towards this end, Google took the engineering know–how that went into Borg, Google’s own in-house

container orchestration platform, and developed Kubernetes. Kubernetes was made open source in 2014 under the Apache 2.0 license in conjunction with the Cloud Native Computing Foundation.

The Mind-shift Required to Work with Kubernetes

Before developing Kubernetes, Google had two choices: give the developer community something like Borg, their in-house orchestration platform, or have them use virtual machines. Because Google was not enamored with the experience virtual machines gave developers, they went down the Borg route. For this very reason, a lot of the conventional thinking around virtual machines needs to be replaced with a fresh mind set. The essential key concepts behind Kubernetes include:

- It is a declarative platform
  In short, you instruct the platform what you want and to the best of its ability the platform attempts to provide what you have asked for. Every object that can be created, configured and managed in Kubernetes can be specified in YAML. YAML is a language that is ubiquitous in the world of software development when it comes to specifying any type of entity declaratively.
- Infrastructure is abstracted away from the developer
  Consider a virtual machine. Despite the word 'Virtual', it looks in many ways like a physical machine and it comes with many of the maintenance overheads that a physical machine comes with. To spin up a virtual machine, a developer needs to allocate storage and networking resources to the virtual machine. By abstracting away the infrastructure a Kubernetes cluster runs on, Kubernetes allows developers to focus on the one single thing that is most important to them, the application. A point that Brendan Burns makes in "The Future of Kubernetes Is Serverless":

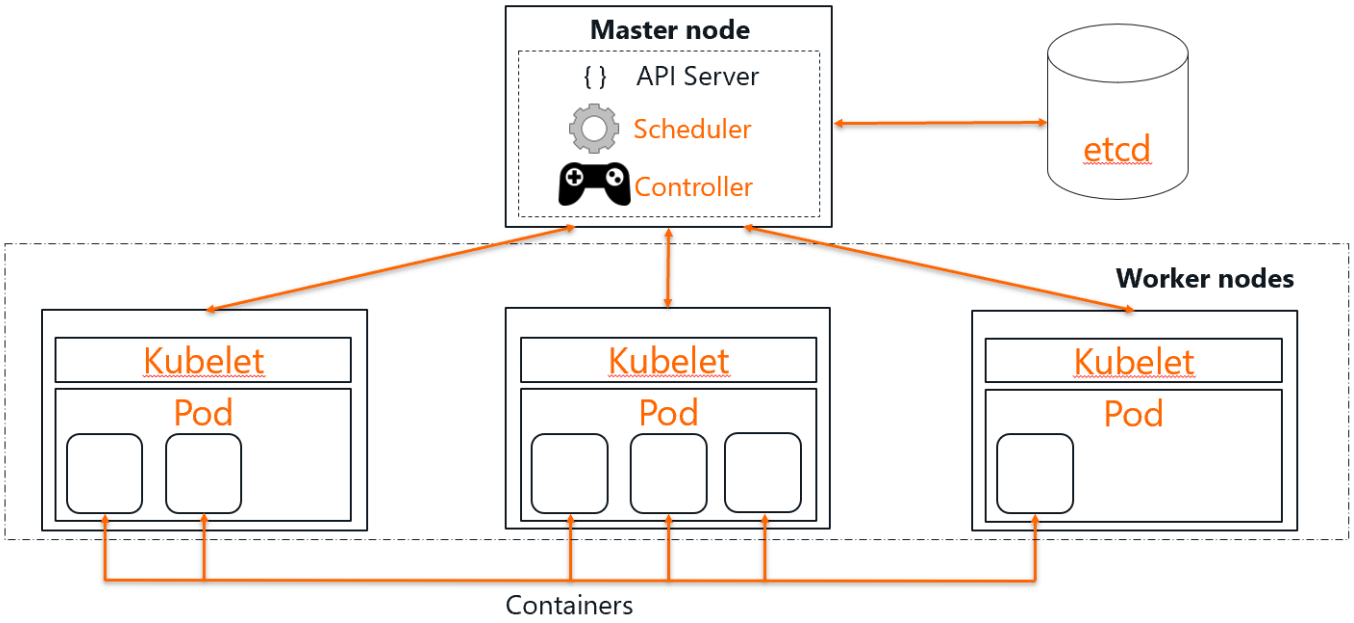
  From the beginning of the container revolution two things have become clear: First, the decoupling of the layers in the technology stack are producing a clean, principled layering of concepts with clear contracts, ownership and responsibility. Second, the introduction of these layers has enabled developers to focus their attention exclusively on the thing that matters to them — the application.
- The Data Center is the Computer
  When deploying applications to a Kubernetes cluster, the concept of individual servers is abstracted away from platform's users. The nodes that make up a Kubernetes cluster will always reside on servers of some sort. However, when using a Kubernetes cluster, the notion of having to logon to specific servers or machines in order to leverage the cluster in any way shape or form does not exist. A great explanation of this concept was given by Kelsey Hightower in the Kubernetes for Sysadmins presentation he delivered at PuppetConf 2016.
- A Platform for Developers
  Kubernetes is a platform for developers that aims to provide a clean separation of concerns between what developers require in order to deploy applications and the underlying infrastructure it runs on.
- A "Platform for building platforms on"
  As per comments made by Kelsey Hightower at KubeCon North America 2018; "It is a platform for building platform on, it is far from the end game". For this reason, platforms such as Istio, a routing mesh for microservices and KNative, a serverless platform, have been built on top of Kubernetes.

The Anatomy of a Kubernetes Cluster

A Kubernetes cluster comprises of two different types of node:

- **master nodes** embody the clusters control plane,
- **worker nodes** service application workloads.

The control plane must always run on Linux, however, the worker nodes can run on Linux or Windows. Cluster state is stored in a simple key value store called etcd which originates from CoreOs. A Kubelet, an entity that is essentially a Kubernetes agent, runs on each worker node. Containers run inside pods. A pod is the unit of scheduling on a Kubernetes cluster. Containers in the same pod are run on the same node.



Master node(s) also furnish the cluster API server, this is RESTful endpoint by which objects can be created, managed and interrogated.

Storage

Applications that run on Kubernetes can be stateless or stateful. State**less** applications use stateless containers, the classic example being something such as Nginx. State**ful** application use containers that persist and manage state, things such as mongodb, redis or any kind of database, NoSQL or otherwise. This is in stark contrast to a virtual machine, which always requires at least one disk.

The touch point for application storage in a Kubernetes cluster is the volume and volumes are always associated to pods. To consume persistent storage and make it available to a pod, persistent volume claims are specified. Two options exist for how space is allocated on actual physical storage devices:

- **Automated provisioning**
  When a persistent volume claim is created, a corresponding persistent volume is created, this is an entity which maps directly to a LUN or volume on the storage device. The association of a persistent volume claim with a persistent volume is

known as "Persistent volume claim binding". This means of storage provisioning provides developers with the most cloud-like storage-as-a-service experience possible.

- Manual provisioning
  Under this storage provisioning scheme, persistent volumes are created manually prior to any persistent volumes claims being created.

Continuously destroying and re-creating storage is not a good practice. Therefore, when a stateful pod is re-scheduled to run on a different node, the Kubelet unmounts any volumes associated with the pod prior to its move, and once the pod has moved, the Kubelet on the destination node re-mounts the volumes.

Some YAML Paints A Thousand Words

To illustrate some of the concepts introduced in this blog, let's look at the YAML required to create a SQL Server instance. In order for this example to work, an object of type secret needs to be created first via this kubectl command:
[crayon-64278f5ed8fcb058991458/]
The following YAML can then be placed in a file, deployment.yaml for example, and applied as follows:
[crayon-64278f5ed8fd6919178452/]
[crayon-64278f5ed8fd7468955087/]
The above excerpt creates three objects:

1. Deployment
   This is the most common means of deploying an application to a Kubernetes cluster. In the excerpt it packages up:

- The containers used in the pods
- A replica set
- The environment required to spin the containerized instance up, found in the section labelled env:
- A password from a Kubernetes object of type secret with the name mssql, i.e. the object that was created by the first kubectl command.
- The volume used by the SQL Server instance pod
- The persistent volume claim that underpins the volume
- Service
  For any application to provide any value, a means of communicating with it from the outside world is required, which is where services come into play. Notice the mssql label in the declaration of the deployment? the selector in the service declaration states that all pods labelled mssql are part of the service. Port 1433 inside the pod is maps to port 1433 on the node and the access to the service is provided via a load balancer endpoint.
- Persistent Volume Claim
  This is the amount of physical storage required by the volume, 8GB in this example allocated from the pure-block StorageClass. Simply put, a persistent volume claim is the 'Bridge' between a pod's volume(s) and a persistent volume that manifests as a volume or LUN on the storage device. Two storage classes represent Pure's two main storage platforms:

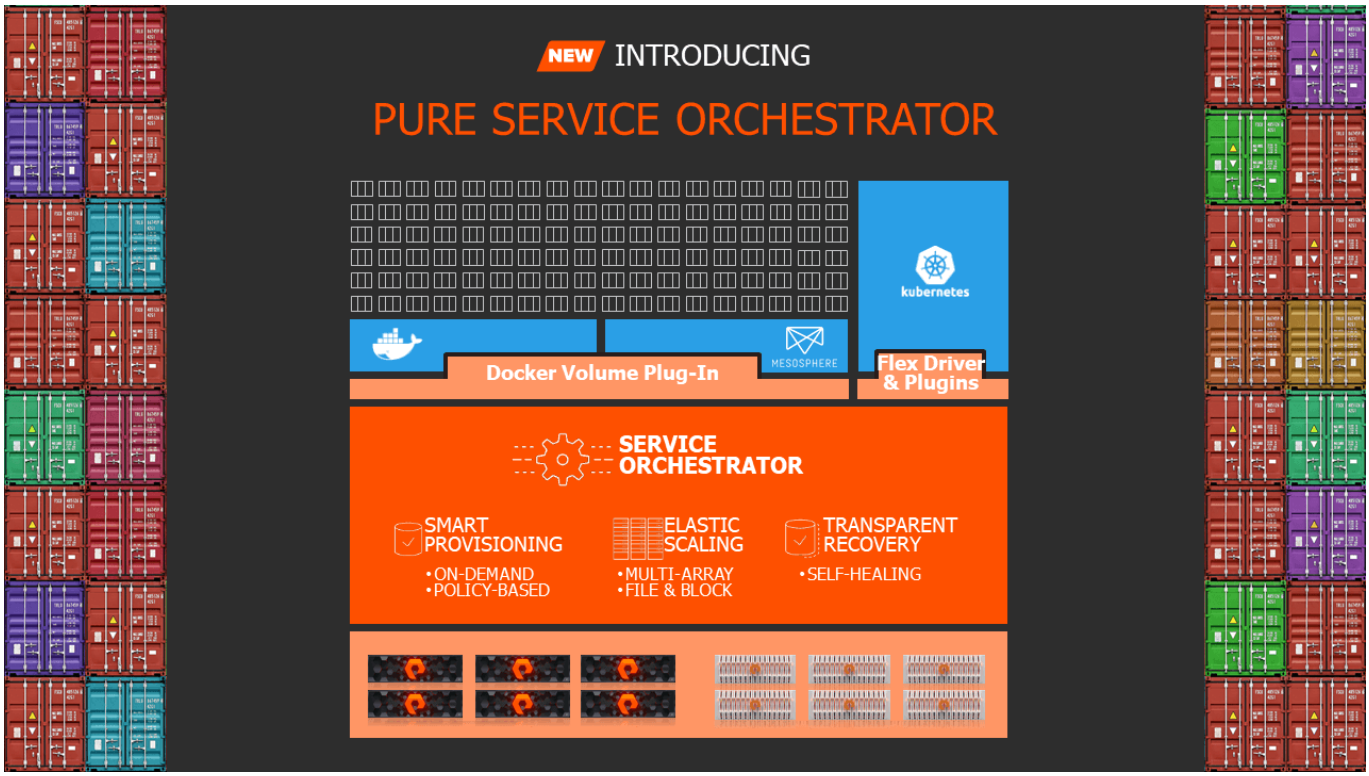
- pure-block
  Use this storage class for allocating storage from Pure's block platform: FlashArray. pure-block is the best choice for applications which are latency sensitive with block oriented storage

requirements; typically relational databases.

- pure-file
  Use this storage class for storage to be allocated from Pure's file and object storage platform: FlashBlade™. Choose this class for applications that are IO bandwidth sensitive in nature.

Introducing Pure Service Orchestrator™

Kubernetes integration into Pure storage platforms is provided by the Pure Service Orchestrator:



In keeping with the Pure mantra of simplicity, the installation and configuration of Pure Service Orchestrator requires just two simple steps:

- The creation of a config map, the most important contents of which are array endpoints (IP addresses) and access tokens,
- Installation of a Helm chart.

With Pure Service Orchestrator, a Kubernetes cluster's storage can be scale out to the orders of petabytes across the industry's leading block and object/file storage platforms. Not only that, but Pure Service Orchestrator intelligently determines where best to create persistent volumes based on a number of items, including the free capacity, performance and health of each storage device.

Why Choose Pure For Your Kubernetes Storage Requirements ?

Of all the applications that run against FlashArray, SQL Server is one of the most popular. The storage consumed by mission-critical SQL Server databases on FlashArray™ systems across the world is of the

order of petabytes. Pure brings FlashArray's qualities and features enjoyed by SQL Server running on bare metal and virtual machines to the the world of containers and Kubernetes, including:

- ability to handle mixed workloads,
- 6 x 9s of availability,
- comprehensive set of always-on data services, including advanced de-duplication and compression, always on at rest encryption and snapshots,
- incredible simplicity and ease of management.

Pure is here to help with your SQL Server journey into the brave new world of containers and Kubernetes. The platforms for running SQL Server on may change and evolve, but the qualities that Pure customers love and enjoy will steadfastly remain the same !.