

# Deploying SQL Server in Kubernetes

**Anthony E. Nocentino**  
[aen@centinosystems.com](mailto:aen@centinosystems.com)



# Anthony E. Nocentino

- **Consultant and Trainer**
- **Founder and President of Centino Systems**
  - Specialize in system architecture and performance
  - Masters Computer Science
  - Microsoft MVP - Data Platform - 2017 - 2020
  - Linux Foundation Certified Engineer
  - Friend of Redgate - 2015-2020
- **email:** [aen@centinosystems.com](mailto:aen@centinosystems.com)
- **Twitter:** @nocentino
- **Blog:** [www.centinosystems.com/blog](http://www.centinosystems.com/blog)
- **Pluralsight Author:** [www.pluralsight.com](http://www.pluralsight.com)



# Agenda

- **Deploying SQL Server in Kubernetes**
  - Data Persistency and Storage in Kubernetes
  - Pod Configuration and Running SQL Server in a Pod
  - Disk and Resource Configurations
  - Backups
  - The ~~Future~~ **present** of SQL Server and Kubernetes

# Kubernetes 101

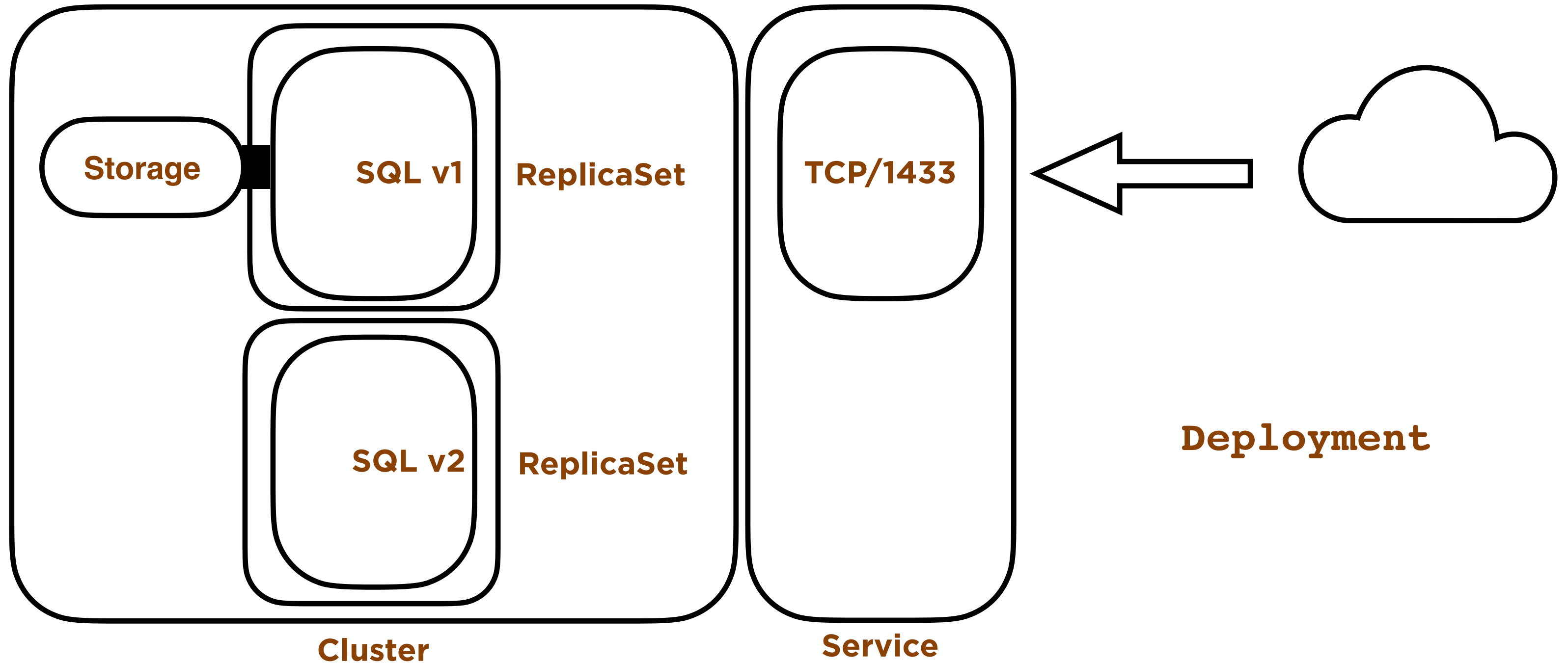
- Container Orchestrator
- Pods are Container Based Applications
- A Cluster is a Collection of Compute Resources
- Infrastructure Abstraction
- Declarative Configuration in Code
- Desired State



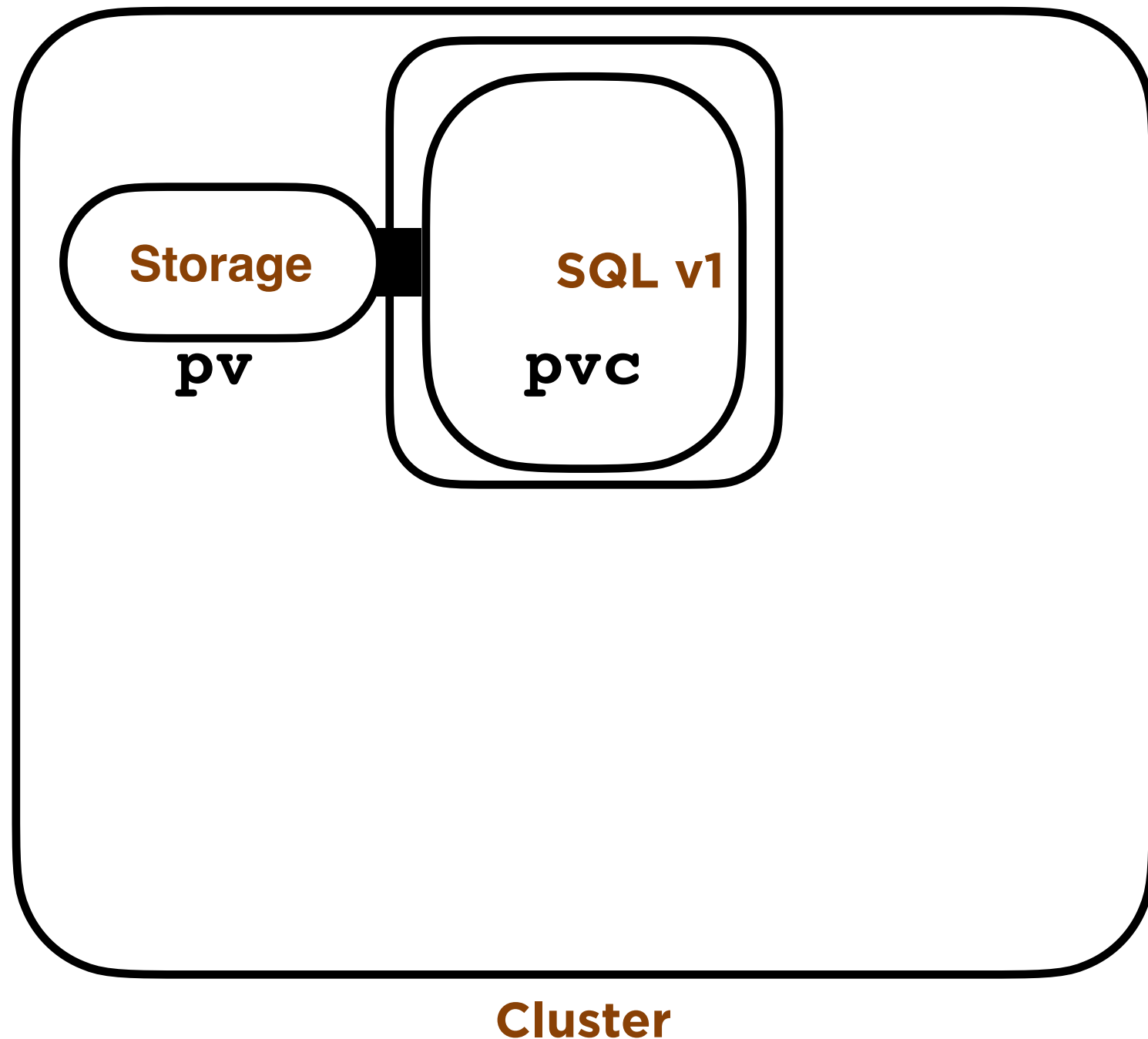
# Running SQL Server in Kubernetes

- A Pod goes back its initial state each time it's deployed
- **State** - where do we store data?
- **Configuration** - how do we configure SQL Server?

# Decoupling Data and Computation



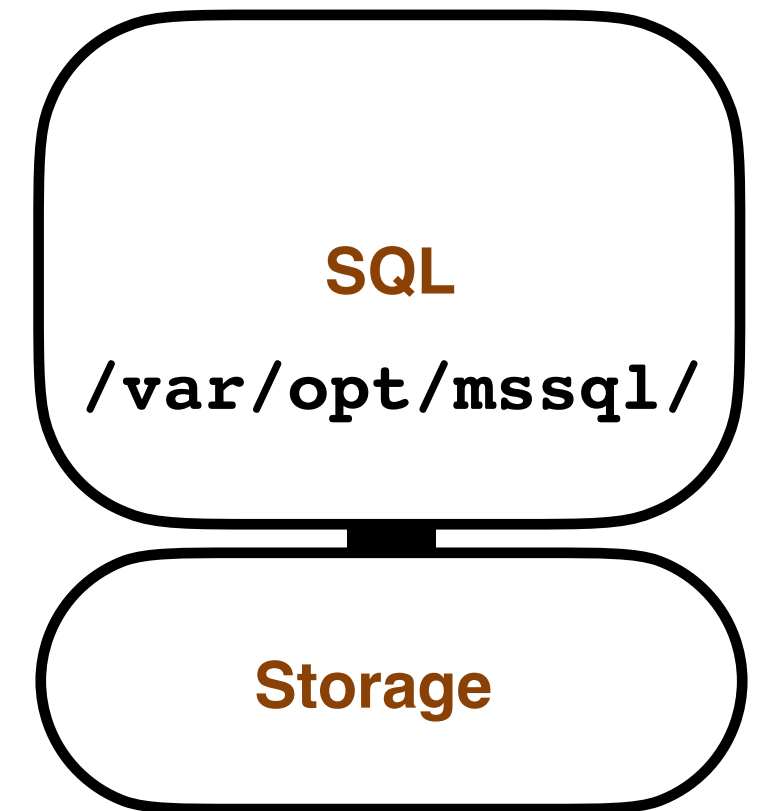
# Storage in Kubernetes



- **Persistent Volumes (pv)**
  - Administrator defined storage
  - iSCSI, NFS, FC, AzureDisk...many more
- **Persistent Volume Claims (pvc)**
  - The Pod “claims” the **pvc**
  - The **pvc** is mapped to the **pv** by k8s
  - Decouples the Pod and the storage

# Data Persistency in SQL Server in K8S

- Define **Persistent Volumes/Persistent Volume Claims**
  - Instance directory (error log, default trace, etc..)
    - **`/var/opt/mssql/`**
  - User Database default directory
    - **`/var/opt/mssql/data`**





# Defining Persistent Volumes and Persistent Volume Claims

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-data
  labels:
    disk: data
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 172.16.94.5
    path: "/export/volumes/sql/data"
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-data
spec:
  selector:
    matchLabels:
      disk: data
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

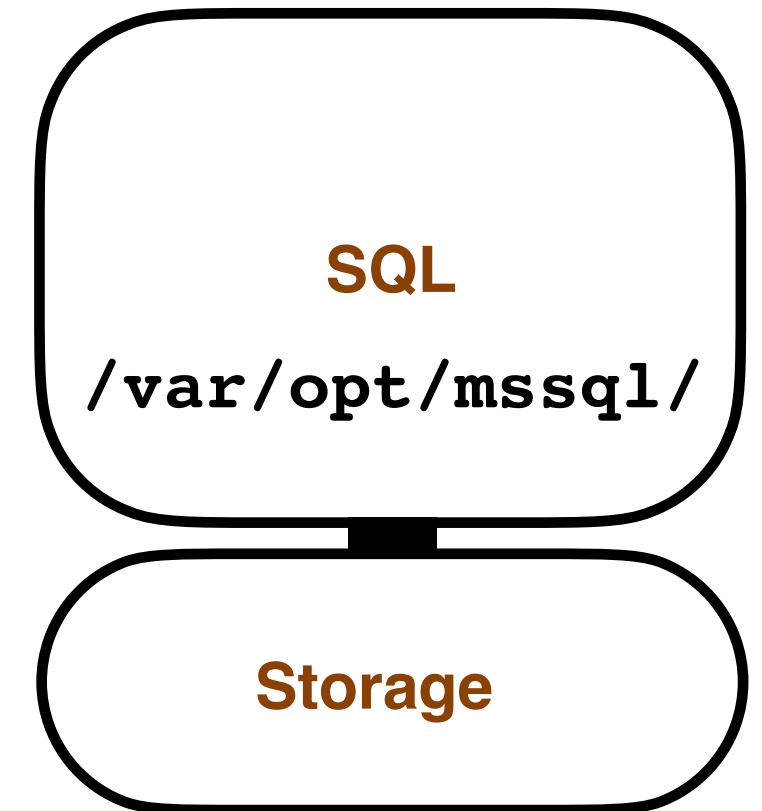
# Configuring SQL Server in a Pod

- In our Pod configuration we define **Environment Variables**
  - Used at initial startup to configure the SQL Instance
    - **ACCEPT\_EULA**
    - **MSSQL\_SA\_PASSWORD**
      - Stored in the cluster as a **Secret**
  - Pods go back their initial state of the container image on creation
  - But some settings are persisted in master, right...yep!

<https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-configure-environment-variables>

# Running SQL Server in a Pod (con't)

- In our Pod configuration define our storage configuration (**pvc**)
- Initial Pod deployment
  - If there's no system databases in the default data directory...
    - **/var/opt/mssql/data**
  - They're copied into the default data directory from the SFPs
- On subsequent Pod deployments the storage is attached into the 'new' Pod
  - Databases are already there
  - Master is read...contains our instance's configuration and state
  - Defined and accessible user databases are brought online



# Define SQL Server in a Pod in YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mssql-deployment
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: mssql
  securityContext:
    fsGroup: 10001
  containers:
    - name: mssql
      image: '.../mssql/server:2019-CU3-ubuntu-16.04'
      ports:
        - containerPort: 1433
      env:
        - name: ACCEPT_EULA
          value: "Y"
        - name: SA_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mssql
              key: SA_PASSWORD
      volumeMounts:
        - name: mssqldb
          mountPath: /var/opt/mssql
  volumes:
    - name: mssqldb
      persistentVolumeClaim:
        claimName: pvc-sql-data
```

# Advanced Disk Topologies for SQL Server

- Define your **Persistent Volumes** and **Persistent Volume Claims**
- Use environment variables to specify default directories on Pod at startup
  - **MSSQL\_DATA\_DIR (/data)**
  - **MSSQL\_LOG\_DIR (/log)**
- New user databases will be created in these locations
- On Pod creation
  - All **PV/PVCs** will be mounted in the container at the defined locations
  - Master will online the databases

# Resource Management

- Pod level resource management

- CPU and Memory

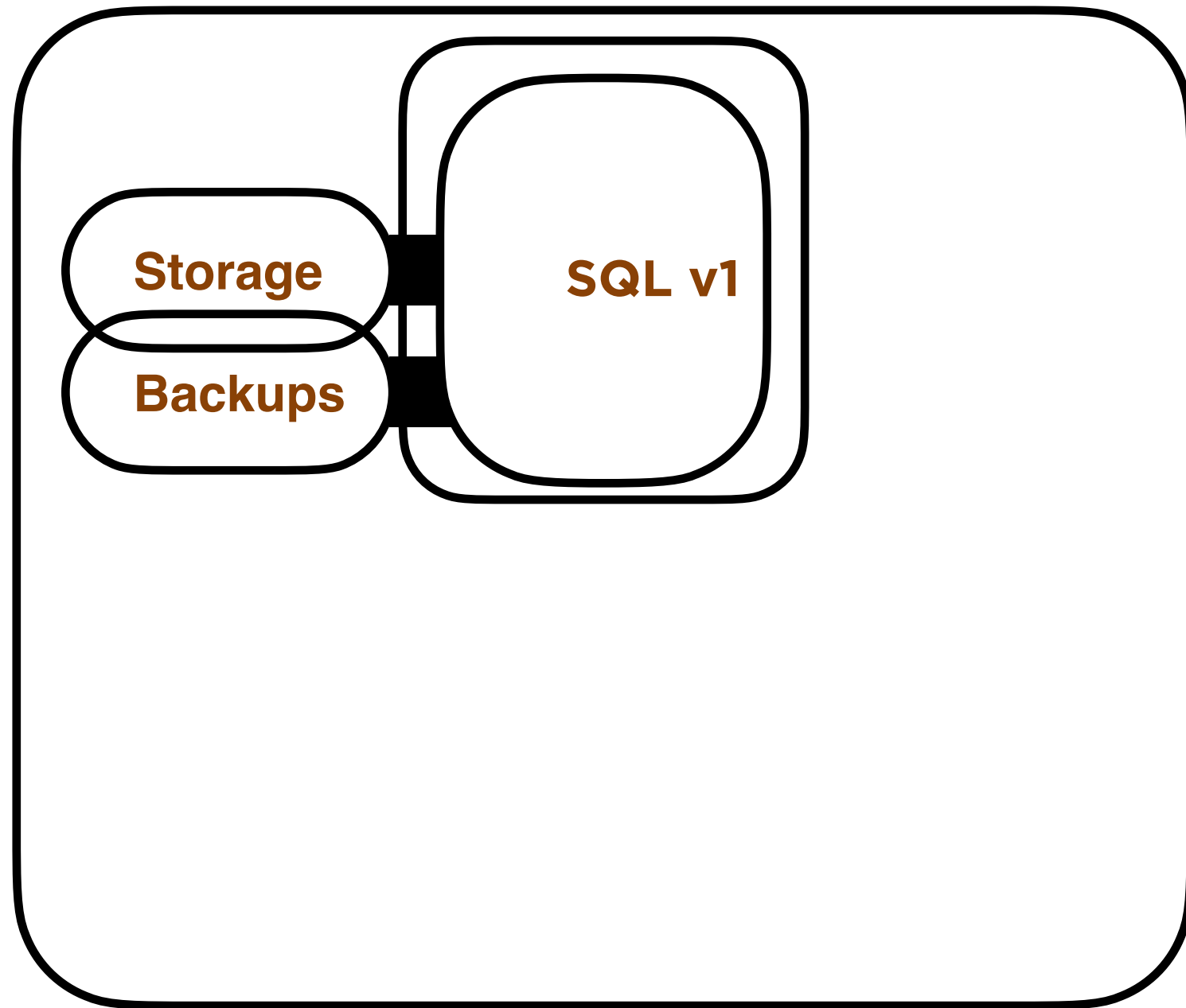
- **requests** - guaranteed
  - **limits** - upper limit
  - No limits by default
- ```
containers:  
- name: mssql  
  image: '.../mssql/server:2019-CU3-ubuntu-16.04'  
  resources  
    requests:  
      cpu: 1  
      memory: 1Gi
```

- Server Instance settings still apply
- Kind of like multi-instance clusters
- Workload is stopped and started when moved between Nodes


URL

AzureFiles

# Backups!



Cluster

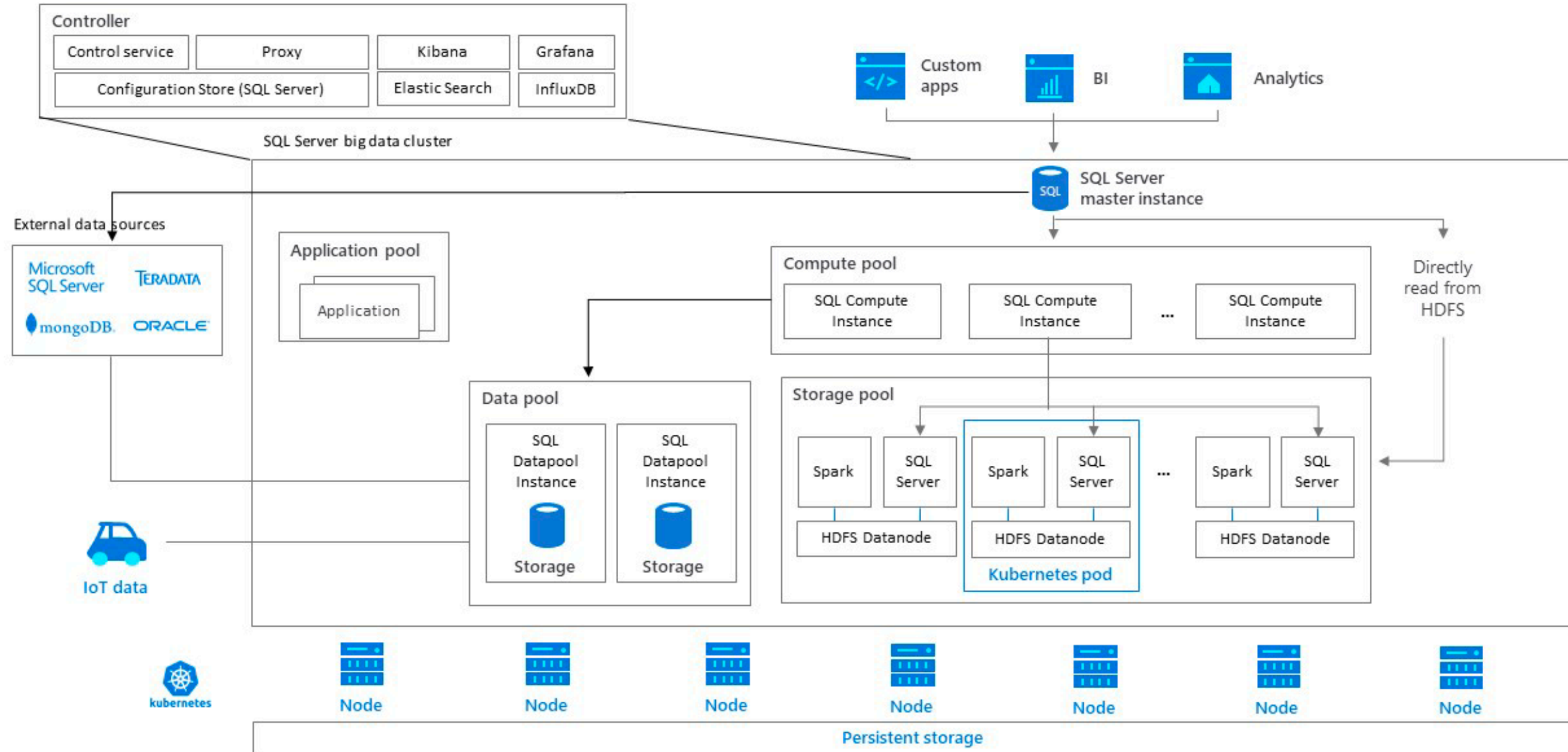
- Persistent Volume (Shared or Dedicated)
  - AzureDisk
  - AzureFile
  - NFS/iSCSI/FC
- To URL
- Drive the backup jobs with normal techniques
  - Ola Hallengren's
  - Maintenance Plans
  -  dbatools

# Demo!

- Deploying SQL Server in a **Deployment** with Persistent Storage
  - Disk Topology
  - Setting Resource Limits
  - High Availability
  - Backing up SQL Server in Kubernetes



# Big Data Clusters



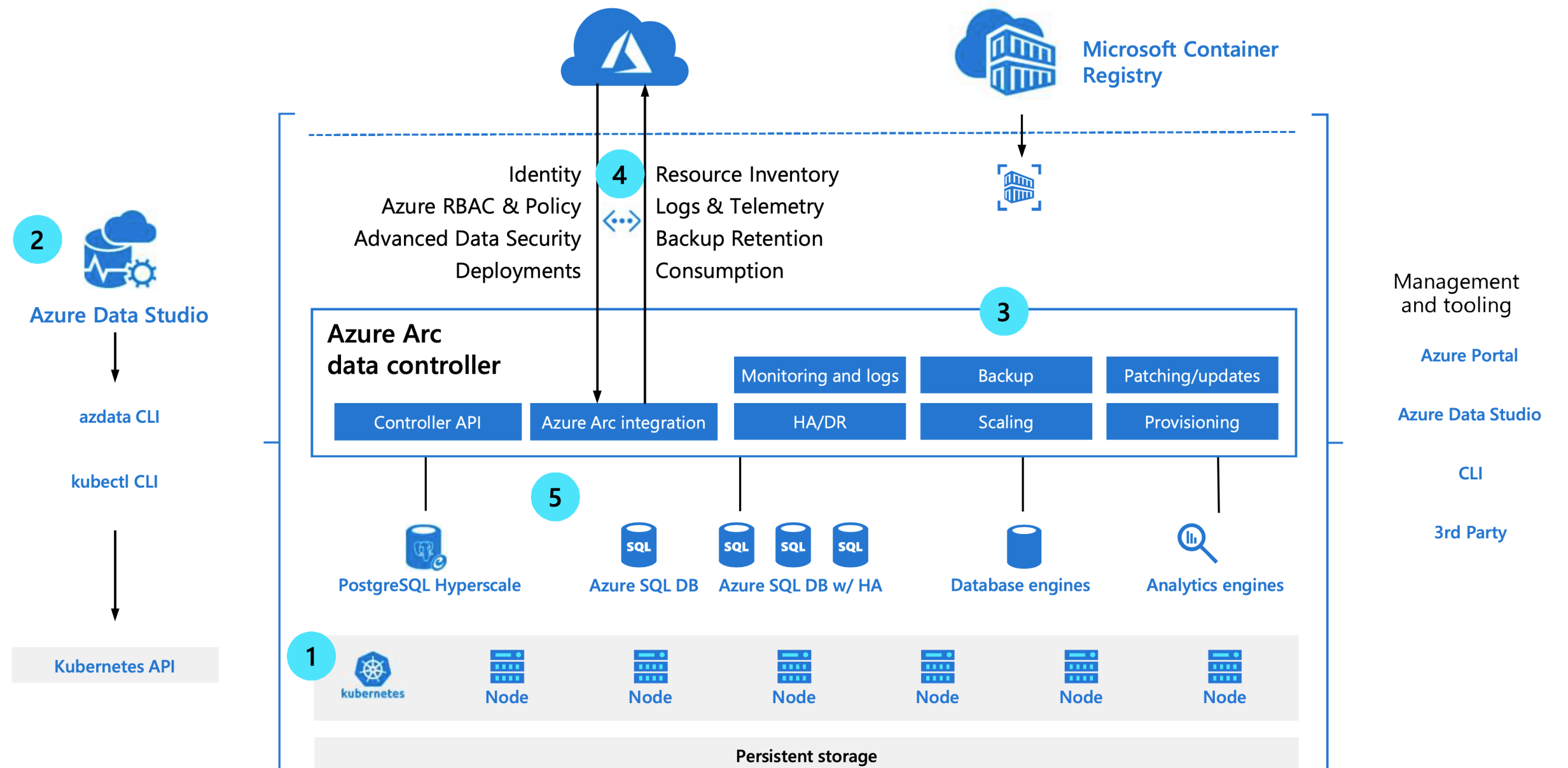
<https://docs.microsoft.com/en-us/sql/big-data-cluster/big-data-cluster-overview?view=sqlallproducts-allversions>

# Azure Arc - Data Services

## How it works: architecture of Azure data services on customer infrastructure

A few steps to get Azure data services in your environment:

- 1 Have Kubernetes on your infrastructure
- 2 Prepare environment with APIs and CLIs
- 3 Install Azure Arc data controller
- 4 Connect to Azure
- 5 Deploy and run Azure data services for your workloads



# Review

- **Deploying SQL Server in Kubernetes**
  - Data Persistency and Storage in Kubernetes
  - Pod Configuration and Running SQL Server in a Pod
  - Disk and Resource Configurations
  - Backups
  - The Future of SQL Server and Kubernetes

# More Resources

- **Docker for Windows/Mac**
- **Managed Service Providers**
  - Azure Kubernetes Service (**AKS**)
    - <https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough>
  - **Elastic Container Service for Kubernetes (EKS)**
    - <https://aws.amazon.com/getting-started/projects/deploy-kubernetes-app-amazon-eks/>
  - **Google Kubernetes Engine (GKE)**
    - <https://cloud.google.com/kubernetes-engine/docs/how-to/>
- **Pluralsight**
  - <https://app.pluralsight.com/profile/author/anthony-nocentino>

# Thank You For Your Time!

[@TheDataGeeks](#)

[@SQLServerGeeks](#)



[www.YouTube.com/SQLServerGeeks](http://www.YouTube.com/SQLServerGeeks)

 **DataPlatformGeeks**  
Virtual Symposium

[www.DataPlatformGeeks.com](http://www.DataPlatformGeeks.com) | [www.SQLServerGeeks.com](http://www.SQLServerGeeks.com)

---

[@TheDataGeeks](#) | [@SQLServerGeeks](#) | [admin@DataPlatformGeeks.com](mailto:admin@DataPlatformGeeks.com)

