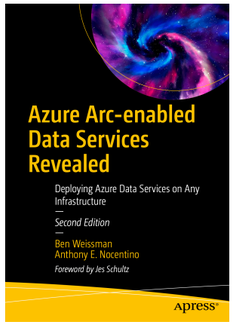
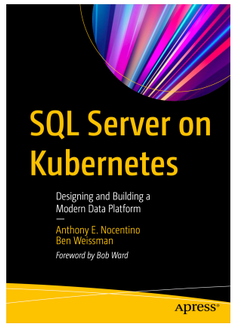
Inside Kubernetes Architecture Fundamentals

Anthony E. Nocentino
anocentino@purestorage.com

Anthony E. Nocentino

- **Principal Field Solution Architect @ Pure Storage**
  - Specialize in system architecture and performance
  - Masters Computer Science
- **email**: anocentino@purestorage.com
- **Twitter**: @nocentino
- **Blog**: www.nocentino.com
- **GitHub:** https://github.com/nocentino/
- **Pluralsight Author**: www.pluralsight.com
- **Founding Organizer of EightKB** - www.eightkb.online





Agenda

- What is Kubernetes
- Benefits of Using Kubernetes
- Kubernetes API Objects
- Exploring Kubernetes Architecture
- Deploying Applications
- Deploying SQL Server

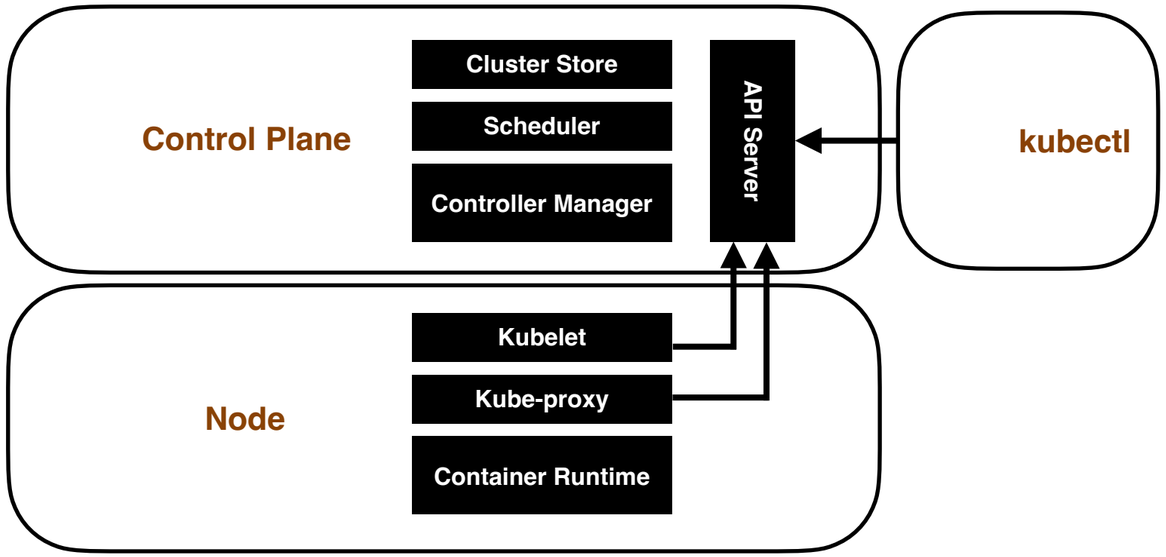
What is Kubernetes?

- Container Orchestrator
- Pods are Container Based Applications
- Infrastructure Abstraction
- Desired State
- Declarative Configuration in Code

Kubernetes Benefits

- Workload placement
- Managing state, starting things up and keeping things up
- Networking and Services
- Load balancing services
- Persistent storage
- Declarative model

Exploring Kubernetes Architecture



Kubernetes API

- **API Objects** - Represent resources in your system
  - **Pods** - your container based applications
  - **Controllers** - maintain desired state
  - **Services** - persistent access to your apps
  - **Storage** - persistent storage for your data
  - …and more

Pods

- One or more containers
- It’s your application
- The most basic unit of work
- Unit of scheduling
- Ephemeral - no Pod is ever “redeployed”

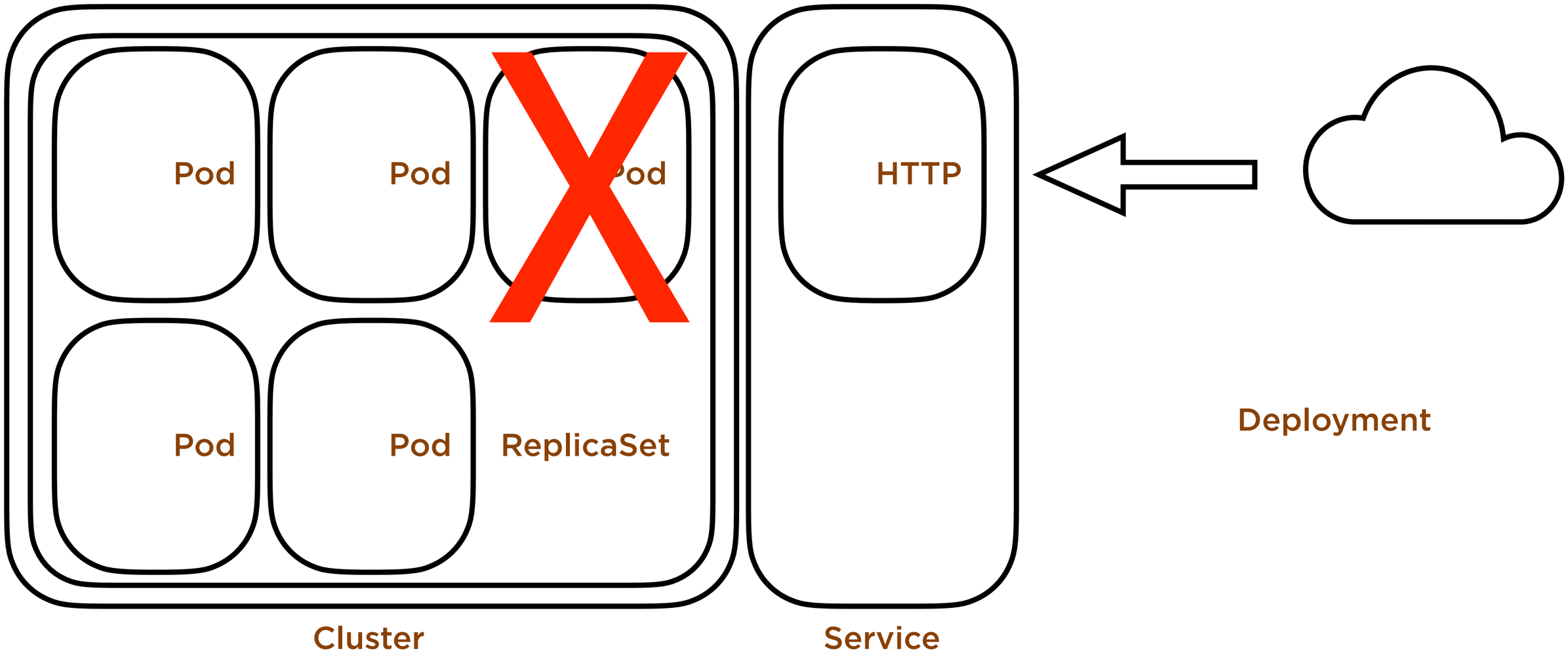
Controllers

- Create and manage Pods for you
- Define your desired state
- Respond to Pod State and Health
- **`ReplicaSet`**
- **`Deployment`**

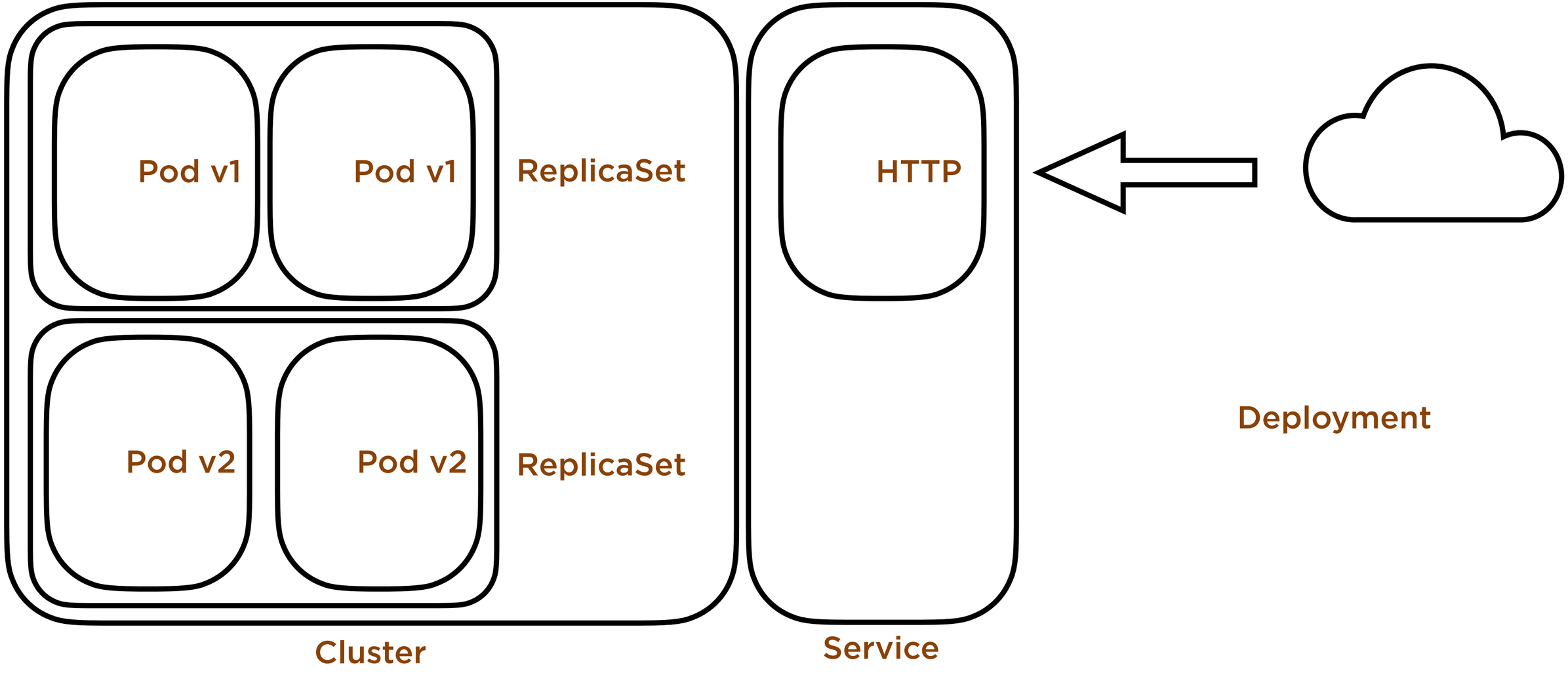
Services

- Adds persistency to our ephemeral world
- Networking abstraction for Pod access
- IP and DNS name for the service
- Load balancing
- Deployed Pods automatically updated
- Scaled by adding/removing Pods

Services and ReplicaSets



Controller Operations - `Deployment`



Deploying Applications

- Imperative
- Declarative
- YAML and JSON

Declarative Deployment - Manifests

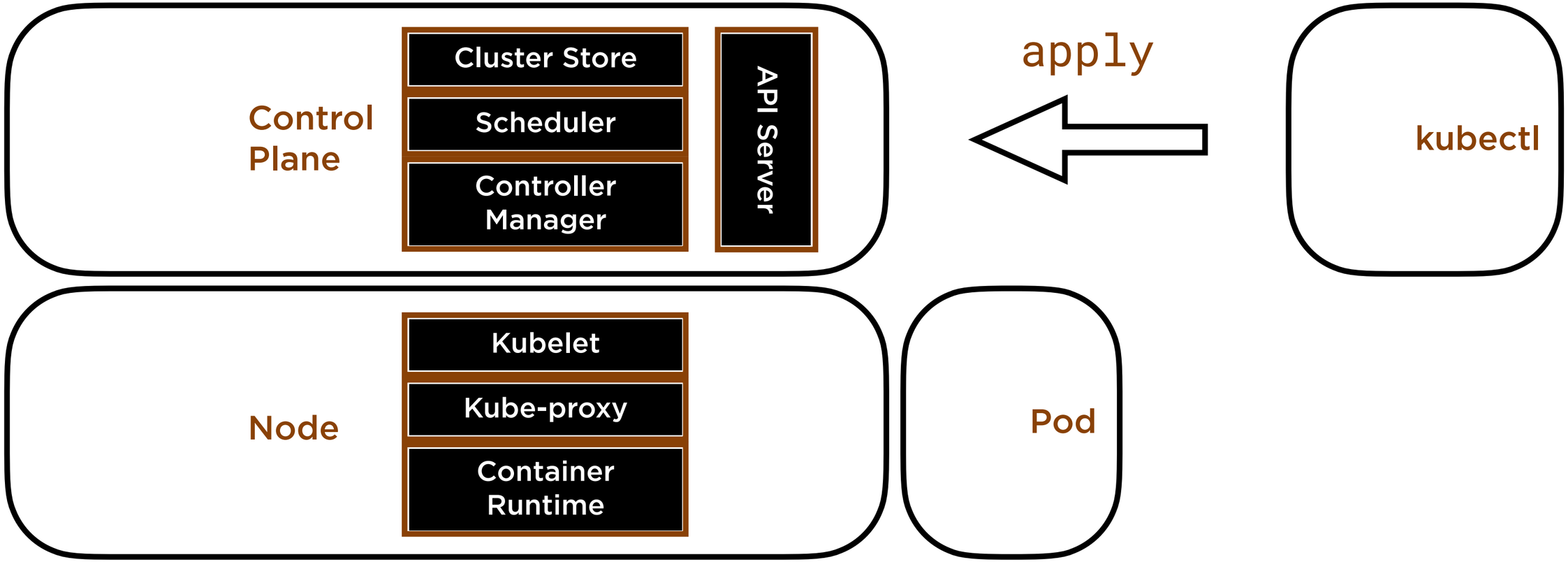
Deployment

Pod Template

```yaml
apiVersion: app/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - image: gcr.io/google-samples/hello-app:1.0
        name: hello-app
```

```
kubectl apply -f deployment.yaml
```

Application Deployment Process



Demo!

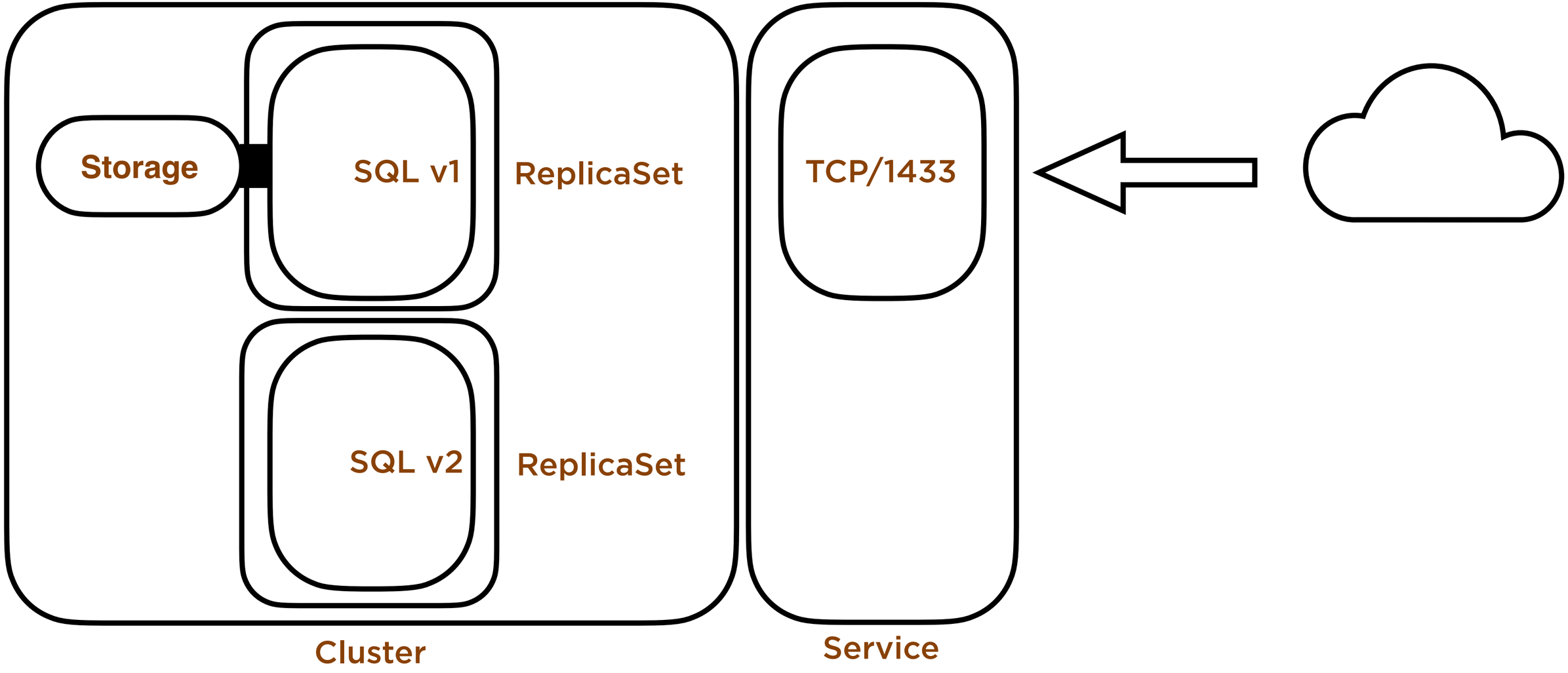
- Imperatively deploying a web application
- Accessing Services within a Cluster
- Declaratively deploying a web application
- Advanced Deployment operations

Running SQL Server in Kubernetes

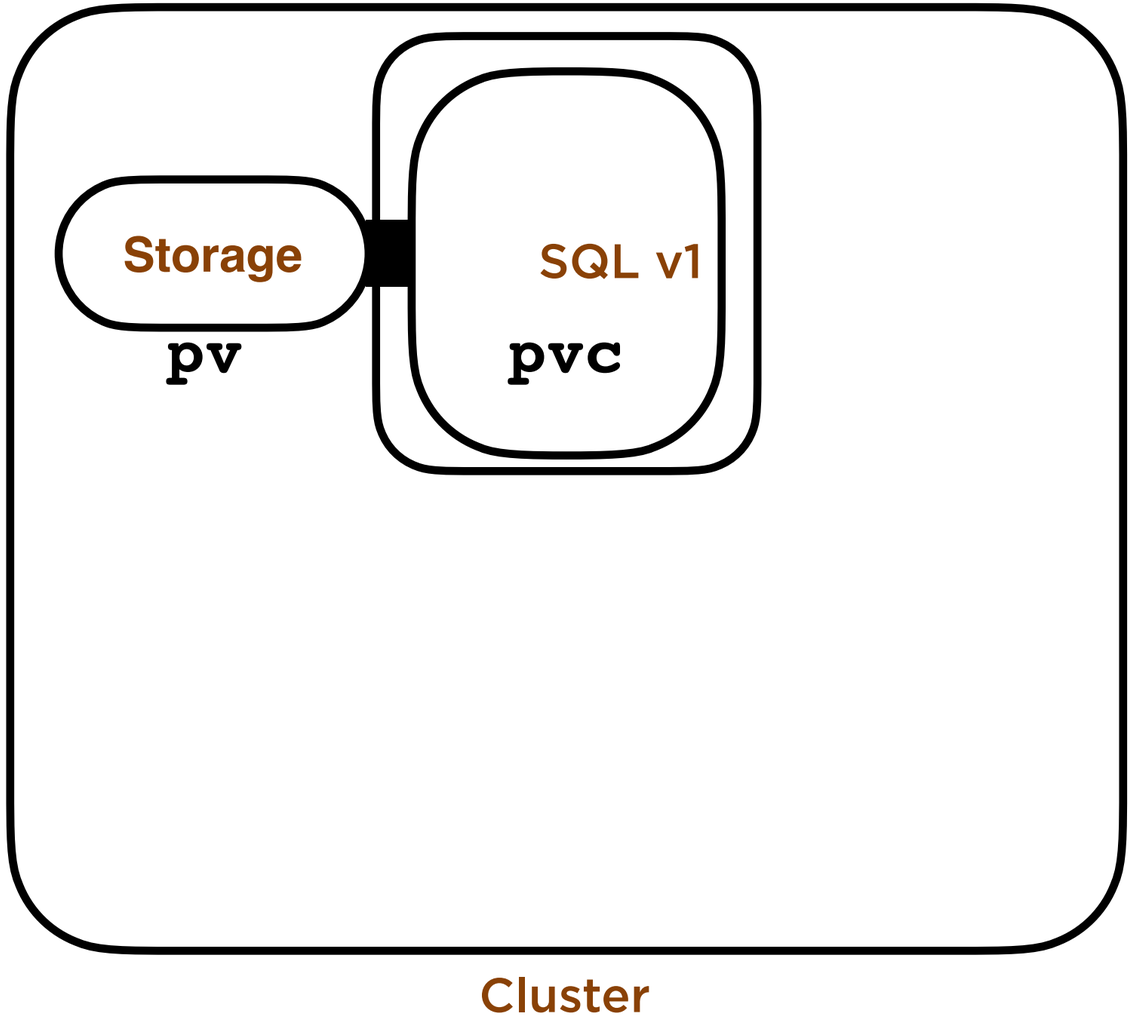
- A Pod goes back to its initial state each time it’s deployed

- **State** - where do we store data?
- **Configuration** - how do we configure SQL Server?

Decoupling Data and Computation



Storage in Kubernetes



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

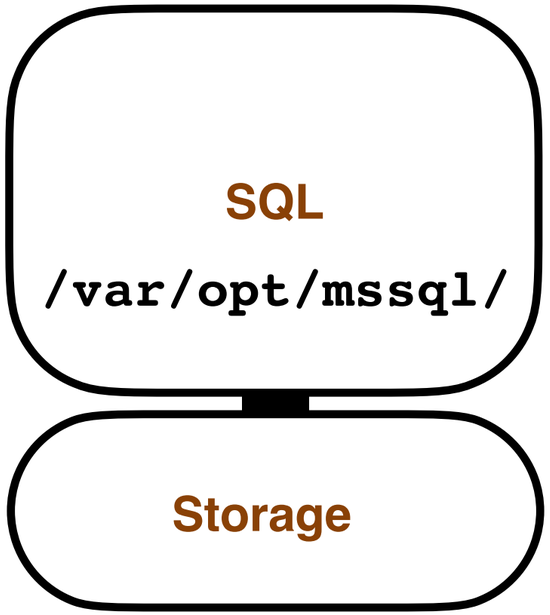
Defining Persistent Volumes and Persistent Volume Claims

```yaml
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"
```

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

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`**



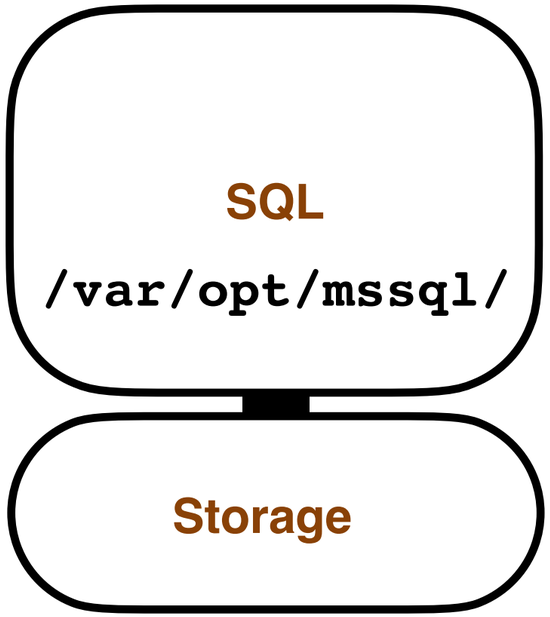
Running SQL Server in a Pod

- In our Pod configuration we define **Environment Variables**
  - Used at initial startup to configure the SQL Instance
    - **`ACCEPT_EULA`**
    - **`SA_PASSWORD`**
      - Stored in the cluster as a **`Secret`** (hashed, not encrypted)
  - Pods go back its initial state of the container image on creation

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

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mssql-deploymen
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
        app: mssql
spec:
  hostname:
    sql01
  securityContext:
    fsGroup: 10001
  containers:
  - name: mssql
    image: '…/mssql/server:2019-CU13-ubuntu-20.04'
    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

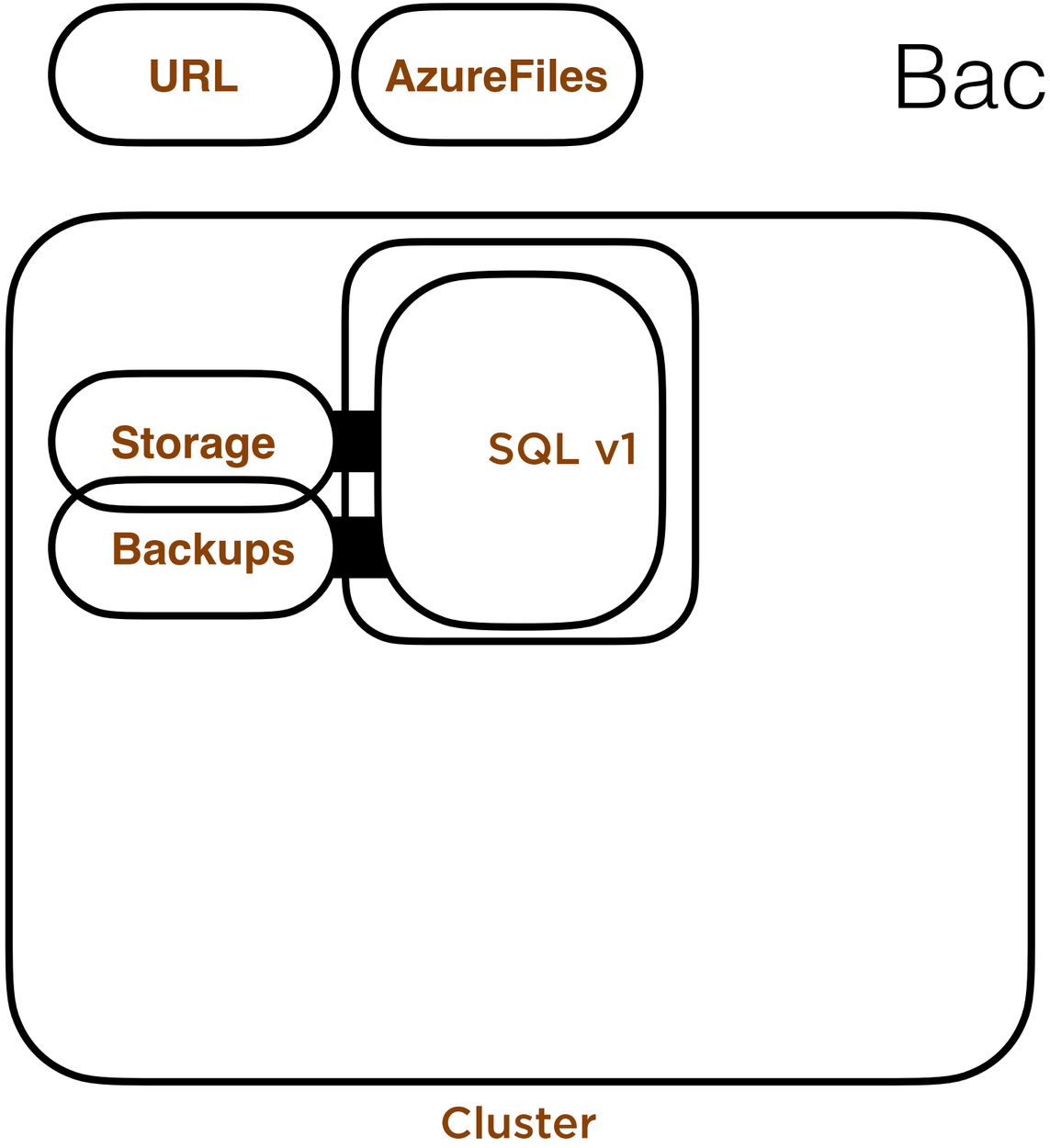
- Define your Persistent Volumes and Persistent Volume Claims
- Use environment variables to specify default directories on Pod at startup
  - **`MSSQL_DATA_DIR (/data)`**
  - **`MSSQL_LOG_DIR (/log)`**
- New user databases will be created in these locations
- On Pod creation
  - All **`PV/PVCs`** will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

http://www.centinosystems.com/blog/sql/data-persistency-and-advanced-sql-server-disk-topologies-in-kubernetes/

Resource Management

- Resource management can happen at the Pod and Namespace levels
  - CPU and Memory
    - **`requests`** - guaranteed
    - **`limits`** - upper limit
  - No limits by default
  - You absolutely should set
  - Server Instance settings still apply

```yaml
containers:
- name: mssql
  image: '…/server:2019-CU13-ubuntu-20.04'
  resources:
    requests:
      cpu: 1
      memory: 1Gi
    limits:
      cpu: 1
      memory: 8Gi
```



Backups!

- 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
  - Backing up SQL Server in Kubernetes
  - Upgrading SQL Server

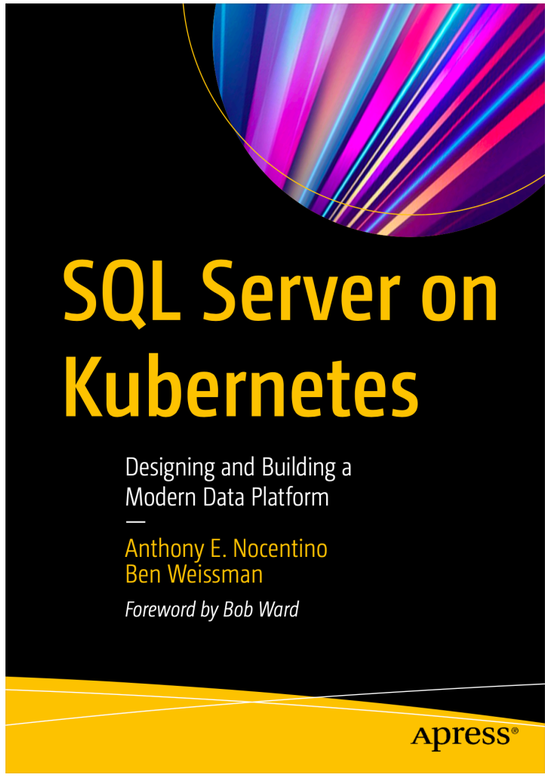
Review

- What is Kubernetes
- Benefits of Using Kubernetes
- Kubernetes API Objects
- Exploring Kubernetes Architecture
- Deploying Applications
- Deploying SQL Server

Need More Data?

- **Contact Me**
  - **Email:** anocentino@purestorage.com
  - **Twitter:** @nocentino
  - **Blog - www.nocentino.com**
  - **GitHub - https://github.com/nocentino/Presentations**

- **Pluralsight**
  - Linux
  - Kubernetes
  - Azure
  - Hit me up for free access to this content



Thank You!