DEPLOYMENT OF DATERA ELASTIC DATA FABRIC WITH KUBERNETES 1.5

Best Practices for Stateful Application Deployment

DEPLOYMENT GUIDE
VERSION 1.0

DATERA

TABLE OF CONTENTS

Datera Overview

Datera Elastic Data Fabric (EDF) is next-generation elastic block storage deployed on industry-standard x86 servers for enterprise and service provider clouds and shared infrastructures. The Datera EDF takes datacenter automation and efficiency to a whole new level, delivering continuous intelligent infrastructure at transformational speed, agility and economics.

Datera EDF provides application-aware, multi-tenant storage that scales near-linearly with performance and capacity with nodes. The system runs on heterogeneous x86 servers (Hybrid Flash Nodes, and All-Flash Nodes) delivering heterogeneous hyper-scale infrastructure that is completely self-aware, self-adaptive, and self-optimizing. The Datera EDF provides a RESTful API that makes the infrastructure programmable and automatable.

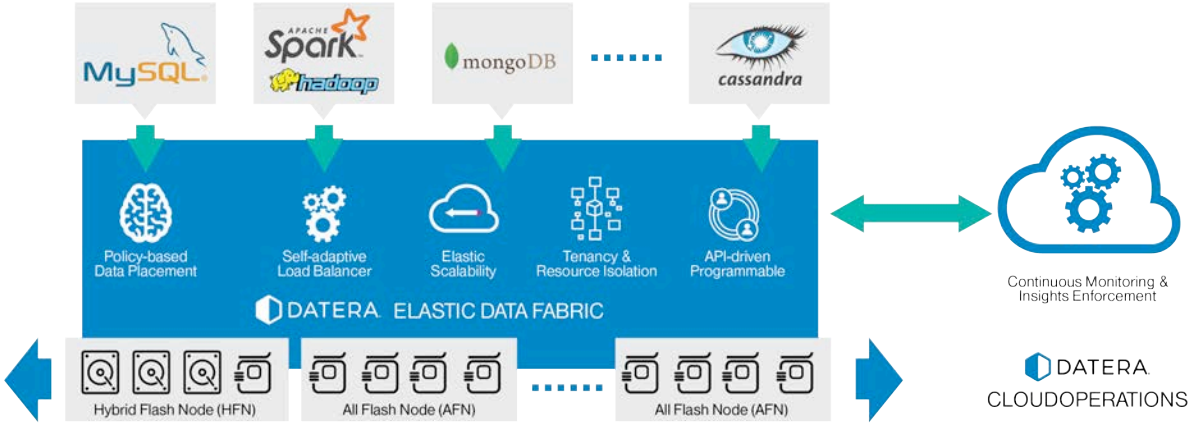

Figure 1: Datera Elastic Data Fabric

Business Benefits of the Datera Platform

Datera architecture optimizes the economic model by using a large fraction of the node capacity with lower cost, high capacity warmer tier and amortizing the cost of the smaller fraction with higher cost, low capacity hotter tier. In addition, the AppTemplates and the policy framework allow configurability of IOPS and bandwidth at an application or volume level. Datera provides a wide range for $/GB and $/IOPS on a per volume basis, thereby providing economical elasticity for a wide variety of cloud applications. And more importantly, automates IT operations on the system through an adaptive data infrastructure.

Given the wide variety of applications hosted in virtual infrastructures, Datera helps match optimize price and performance by truly matching application intent with the right storage. Datera allows for storage to be placed intelligently maintaining the intended service level objectives for any application workload, even in multi-tenant clouds.

- *Infrastructure consolidation*: Supports unified infrastructure for multiple tenants and applications while guaranteeing customized levels of performance and security
- *Self-Managing*: Reduces the need for dedicated storage administrator to manage the cluster and configure and load balance storage
- *Storage efficiency*: With default thin provisioning, zero-overhead on cloning and snapshots, cloud administrators can realize significant effective capacity to used capacity savings
- *Deployment velocity*: Templates empower the developer to rapidly provision applications and can leverage application best practices
- *Programmability*: REST-based API allows for integration to any management or orchestration layer with the multi-tenant roles and privileges and allows customers to integrate with developer tools seamlessly
- *Economics*: Tiering allows for cost optimized $/GB, $/IOPS and $/GB/s given that the working set for development and test workloads is a fraction of the complete data set
- *Infinite scale-out*: Supports the grow-as-you-go model for cloud operators, eliminates fork-lift upgrades, and support for multiple generations of hardware and configurations
- *True multi-tenancy and multi-application support*: Supports unified infrastructure for multiple tenants and applications while guaranteeing customized levels of performance and security including noisy-neighbor fencing
- *Highly available*: Ability to upgrade the hardware and software without service disruption or downtime and without the requirement to go to secondary site or backup resources

Datera Advantage for Containers

Datera Elastic Data Fabric changes the cloud storage playing field and raises the bar far beyond traditional storage solutions capabilities in several areas, including:

1. **Kubernetes Integration:** Datera provides simple installation of the plugin to automatically integrate with Kubernetes and backend storage
2. **Distributed Persistence**: For distributed applications, there is a need for distributed infrastructure layer providing a common operational framework for stateful and stateless workloads. This can provide consistent scale-out access anywhere in the cluster for both ephemeral and persistent storage tiers. In addition, Datera optimizes for distributed access by using container locality, infrastructure failure domains, and resource segregation.
3. **Grow-as-you-go Model**: Start small, and be able to scale fast. Traditional storage is limited in scale by the proprietary hardware frame size and performance, while Datera delivers "heterogeneous COTS scale-out". Software-based system will organically evolve—grow with new hardware, and decommission obsolete old hardware without any disruption. Data gets rebalanced and access optimized for the runtime workloads. No explicit data migration. No forklift upgrades. Ever.
4. **Container Scale and Velocity**: Containers are at least 10x denser than virtual machines. Given the scale and velocity of transactions, the storage infrastructure has to support low

latency, fast operations, and scale of storage objects. Datera provides the speed of provisioning and deployment for containers at scale.

5. **Intent-Defined Container Deployment**: Application specific templates define the I/O characteristics of an application environment. Templates can define broad service classes or specific requirements for each workload in an application environment. Datera maps the intent defined by the app template to the available storage when volumes are created. Each volume is unique, allowing composable, on demand storage infrastructure. Datera system allows for storage to be placed intelligently maintaining the intended services level objectives for any application workload, even in multi-tenant clouds.
6. **DevOps Operations Model**: Datera provides application-driven real-time resource consumption, evolving datacenters from slow infrastructure-centric IT to agile DevOps-centric IT. Application requirements are captured in context-aware policies ("intents") that deeply shape the fabric, accommodating failure domains, network topology, multi-tenancy, workload isolation, etc. An intent-based REST API makes the infrastructure programmable and composable, and allows instantiating entire storage clouds with single API calls (datacenter-as-code). Datera transforms datacenter speed, agility and experience.
7. **Transparent DevOps to IT Ops transition:** Datera allows for application related storage provisioning to be decoupled from the physical infrastructure management. Moreover, customers can seamlessly transition from development to test to IT operations without application configuration changes, but morphing the infrastructure constraints through policy overrides based on application, user or tenant.
8. **Wide Price/Performance aperture:** Given the wide variety of containerized applications, Datera provides the elastic price bands that fit the economic value for particular data. Each volume's requirements are composable on-demand and that articulates storage placement. Datera system allows for storage to be placed intelligently maintaining the intended services level objectives for any application workload, even in multi-tenant clouds.

Contribution to Open Source

Datera is committed to the open source community and has been a contributor to various Linux, Kubernetes, CloudStack and OpenStack projects. In addition, Datera is the primary contributor and maintainer of Linux target mode stack, Linux-IO Target (LIO™)[1] and various IO path modules for the Linux kernel. Datera continues to contribute patches and enhancement to the community. These contributions are widely adopted in industry and have been one of the primary drivers for the emergence of software defined storage market and its use of Linux based storage targets.

In addition, the Datera plugin drivers are available for public access at https://github.com/Datera.

[1] www.linux-iscsi.org Linux-IO Target (LIO™) is the standard open-source SCSI target in Linux. It supports all prevalent storage fabrics, including Fibre Channel, FCoE, IEEE 1394, iSCSI, NVMe-OF, iSER, SRP, USB, vHost, etc.

Datera FlexVolume Driver for StatefulSets[2] in Kubernetes

StatefulSets are intended to be used with stateful applications and distributed systems.

StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.

In the above, stable is synonymous with persistence across Pod (re)scheduling. If an application doesn't require any stable identifiers or ordered deployment, deletion, or scaling, you should deploy your application with a controller that provides a set of stateless replicas. Controllers such as Deployment or ReplicaSet may be better suited to your stateless needs.

[2] StatefulSets is a feature available in Kubernetes 1.4 and later.

Network Preparation

iSCSI uses the same network stack for storage as does the compute layer. Care should be taken to setup the networking in accordance with Datera installation guide. Datera network topology typically looks like the following schematic:

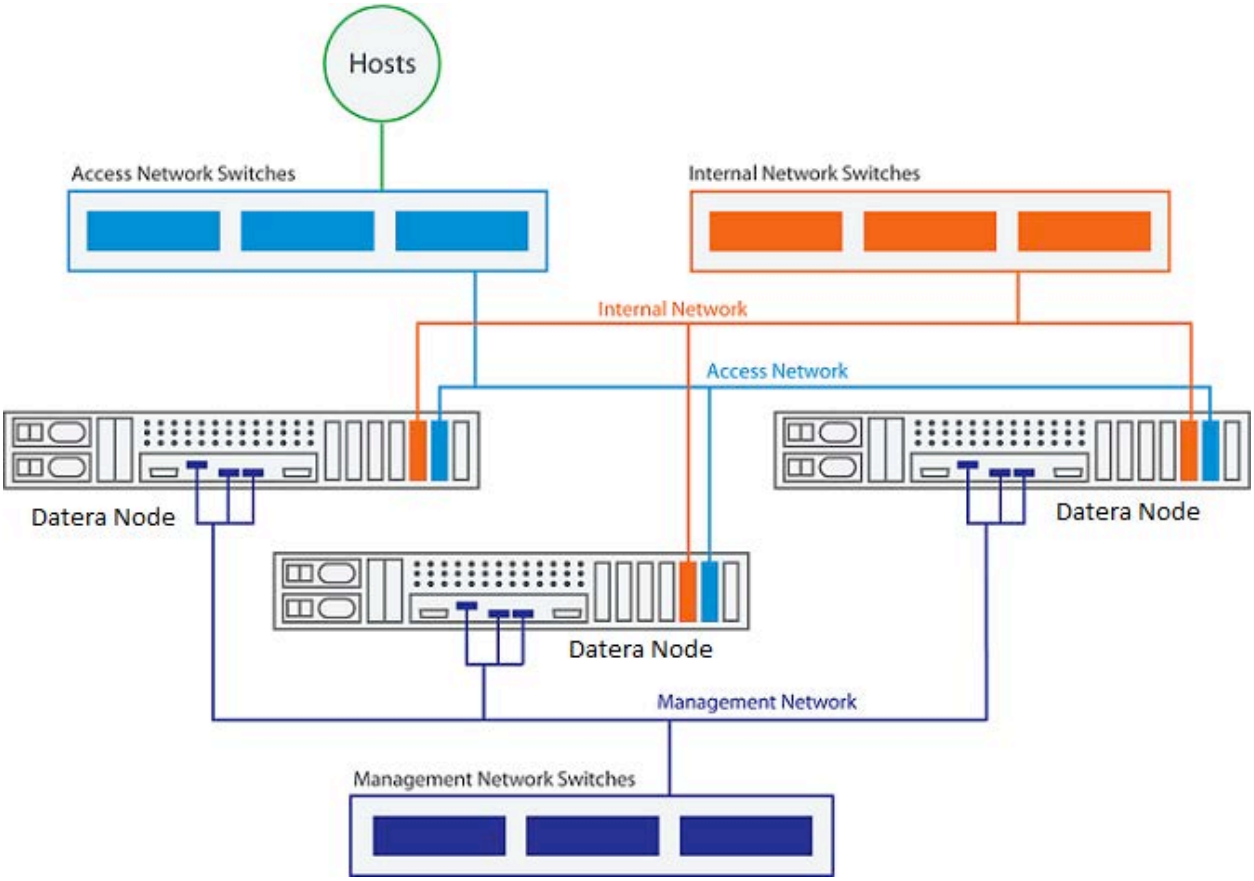

Figure 2. Datera Logical Network Topology

Multiple private networks help with Datera inter-node communications and management. The Access network is the storage network that the client nodes (Nova virtual machine hosts) use to communicate with the storage. This access network is usually dedicated for storage; public access networks are segregated into other VLANs. In planning for Datera networking all these different network segments should be architected and deployed in their own VLANs for optimal storage performance.

Datera Access Network

The Kubernetes minion nodes access the Datera storage through the Access Network iSCSI ports. Each Datera node connects to the Access Network through one or two 10 Gigabit Ethernet (GbE) interfaces. The Access Network must be a traditional IP-based network. The target IP addresses float during node failure, and target port rebalancing activities that occur as part of the load redistribution activities.

The Access Network uses floating IP addresses to present iSCSI LUNs to hosts. The IP address range you provide must at least be equal to the number of nodes in the Datera storage system. However, the more IP addresses available the better the load distribution for the system. Datera recommends at least 32 addresses for small systems with up to 10 nodes, 64 addresses for medium systems up to 20 nodes, and 128-250 addresses for large systems greater than 20 nodes. For more details on the latest recommendations, please refer to the Datera Installation Guide and Datera User's Guide.

The target IP addresses presented by the target iSCSI ports are drawn from the Access Network IP pools. The IP address in the pools are defined by the Access Network IP blocks. You must create at least one IP pool and at least one block of IP addresses to be assigned to Storage Instances. In the user interfaces, the Access Network ports are identified as Access VIP 1 and Access VIP 2.

Configuring Kubernetes with Datera Plugin based on FlexVolume Driver

```
kubectl create –f http://datera.io/assets/files/datera-k8s-
installer.yaml
```

Verify that the Datera installer agents are running on all the Minion nodes

```
root@k8s-controller:~# kubectl get pods –namespace=datera
NAMESPACE     NAME
READY     STATUS    RESTARTS   AGE
datera        datera-installer-agent-0h0zc
1/1       Running   3          17d
datera        datera-installer-agent-5lsds
1/1       Running   3          17d
datera        datera-installer-agent-98brv
1/1       Running   4          18d
datera        datera-installer-agent-9xd1x
1/1       Running   3          17d
datera        datera-installer-agent-f0fxs
1/1       Running   1          18d
datera        datera-installer-agent-fdghn
1/1       Running   3          17d
datera        datera-installer-agent-ttstb
1/1       Running   3          17d
```

Check the Kubernetes Version on the Cluster. In addition, in the case of any minion addition or deletion, the Datera plugin driver will automatically get deployed or terminated on minion without any user intervention. The Datera plugin driver is fully integrated daemon-set in the Kubernetes framework.

```
root@datera-k8s-1:~# kubectl version
Client Version: version.Info{Major:"1", Minor:"5",
GitVersion:"v1.5.1",
GitCommit:"82450d03cb057bab0950214ef122b67c83fb11df",
GitTreeState:"clean", BuildDate:"2016-12-14T00:57:05Z",
GoVersion:"go1.7.4", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"5",
GitVersion:"v1.5.1",
GitCommit:"82450d03cb057bab0950214ef122b67c83fb11df",
GitTreeState:"clean", BuildDate:"2016-12-14T00:52:01Z",
GoVersion:"go1.7.4", Compiler:"gc", Platform:"linux/amd64"}
```

Using StatefulSets[3]

StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.

[3] https://kubernetes.io/docs/concepts/abstractions/controllers/statefulsets/
https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/

- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.

In the above, stable is synonymous with persistence across Pod (re)scheduling. If an application doesn't require any stable identifiers or ordered deployment, deletion, or scaling, you should deploy your application with a controller that provides a set of stateless replicas.

Pod Identity

StatefulSet Pods have a unique identity that is comprised of an ordinal, a stable network identity, and stable storage. The identity sticks to the Pod, regardless of which node it's (re)scheduled on.

Ordinal Index

For a StatefulSet with N replicas, each Pod in the StatefulSet will be assigned an integer ordinal, in the range [0,N), that is unique over the Set.

Stable Network ID

Each Pod in a StatefulSet derives its hostname from the name of the StatefulSet and the ordinal of the Pod. The pattern for the constructed hostname is **$(statefulset name)-$(ordinal)**.

Configuring the Persistent Storage

1. Create the Persistent Volume

```
kubectl create –f datera–pv.yml
persistentvolume "demo1–pv–datera–0" created
```

```
apiVersion: v1
  kind: PersistentVolume
  metadata:
    name: mysql–data–vol–datera–0
    labels:
      volumeId: "demo–vol–0"
  spec:
    capacity:
      storage: 10Gi
    accessModes:
      – ReadWriteOnce
    persistentVolumeReclaimPolicy: Retain
    flexVolume:
      driver: "datera/iscsi"
      fsType: "xfs"
      options:
        volumeID: "demo–pv–datera–0"
        size: "10"
        replica: "3"
        backstoreServer: "172.27.25.101:7717"
        serverUser: "admin"
        serverPasswd: "password"
```

```
root@datera-k8s-1:~/datera-pv# kubectl get pv
NAME                CAPACITY   ACCESSMODES   RECLAIMPOLICY
STATUS      CLAIM     REASON    AGE
demo1-pv-datera-0   10Gi      RWO           Retain
Available                       38s
```

2. Create the Persistent Volume Claim

```
$ kubectl –f datera-pv-claim.yaml

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-vol-datera-0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  selector:
    matchLabels:
      volumeId: "demo-vol-0"

root@datera-k8s-1:~# kubectl create –f datera-pv/datera-pv-
claim.yaml
persistentvolumeclaim " mysql-vol-datera-0" created
root@datera-k8s-1:~# kubectl get pvc
NAME                     STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
mysql-vol-datera-0             Pending
52m
```

```
root@datera-k8s-1:~/datera-pv# kubectl get pv
NAME                        CAPACITY   ACCESSMODES
RECLAIMPOLICY   STATUS    CLAIM                   REASON
AGE
mysql-data-vol-datera-0   10Gi       RWO           Retain
Bound     default/data-mysql-0             28s
```

Start the StatefulSet

```
root@datera-k8s-1:/home/datera# kubectl create –f
http://k8s.io/docs/tutorials/stateful-application/mysql-
statefulset.yaml
statefulset "mysql" created

kubectl get pods –l app=mysql –watch
NAME      READY     STATUS    RESTARTS   AGE
mysql-0   0/2       Pending   0          54s
```

Scaling a StatefulSet

Scaling a StatefulSet refers to increasing or decreasing the number of replicas. This is accomplished by updating the **replicas** field. You can use either **kubectl scale** or **kubectl patch** to scale a Stateful Set.

Deleting StatefulSets

StatefulSet supports both Non-Cascading and Cascading deletion. In a Non-Cascading Delete, the StatefulSet's Pods are not deleted when the Stateful Set is deleted. In a Cascading Delete, both the StatefulSet and its Pods are deleted. One will have to delete the StatefulSet followed by the PersistentVolumeClaim and then PersistentVolume.

Deletion of PersistentVolumeClaims don't delete the PersistentVolumes, and PersistentVolumes are used for mapping to the backend storage volumes. While deletion of PersistentVolumeClaims shouldn't have any impact to data, please use caution.

```
kubectl delete –f http://k8s.io/docs/tutorials/stateful–
application/mysql–statefulset.yaml
```

The reclaim policy for a PersistentVolume tells the cluster what to do with the volume after it has been released of its claim. Currently, volumes can either be Retained, Recycled or Deleted.

Retaining reclaim policy allows for manual reclamation of the resource. When the PersistentVolumeClaim is deleted, the PersistentVolume continues to exists (and the data is preserved), and the volume is considered "released", but however, the volume is not available for another claim.

Recycling reclaim policy performs a basic scrub on the volume and makes it available again for a new claim.

Datera volume plugin supports the Delete reclaim policy, deletion removes both the PersistentVolume object from Kubernetes, as well as deleting the associated storage asset in the Datera backend storage device (external infrastructure).

Conclusion

Datera is a highly scalable platform for Kubernetes environments allowing for horizontal scale in capacity and performance as you go. The seamless integration of Datera driver for the FlexVolume framework for Kubernetes makes it very simple to provision and manage the Datera system. Datera helps makes Kubernetes clouds successful with a wide price/performance band for a wide range of applications, non-disruptive software and hardware upgrades, grow-as-you-go and simplified operations.