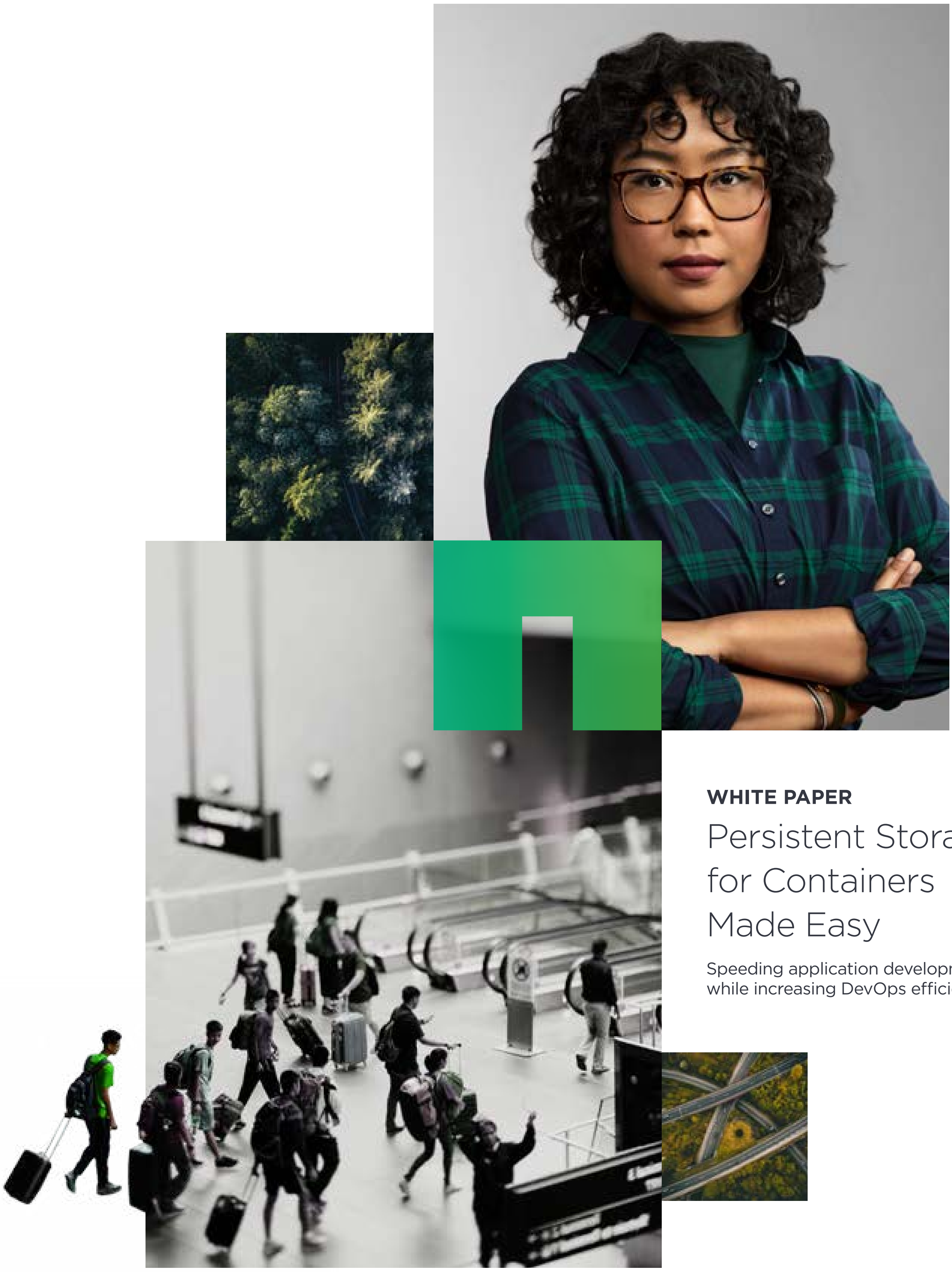


WHITE PAPER

Persistent Storage for Containers Made Easy

Speeding application development while increasing DevOps efficiency

NetApp®

The Challenge: Building DevOps Maturity

Continued innovation. Better, faster product rollouts. More streamlined operations. A growing pool of happy customers.

These are universal goals for many organizations. They are also milestones along the drive toward better business outcomes: greater top-line revenue growth and improved bottom-line profitability.

On the flip side of such drives toward achievement are often the growing pains and frustrations of an organization in the midst of transformation. Application developers—eager to accelerate development efforts that solve customer needs—might find themselves frustrated with ticketing systems and long waits for storage and compute resources. Infrastructure and operations (I&O) teams, in contrast, struggle to keep up with incoming service tickets and with controlling the way infrastructure is being utilized.

This mixture of goals and growth pains is propelling organizations to pursue agile and lean methodologies to drive greater speed and efficiency in their product development. And while agile and lean methodologies have delivered significant benefits, far greater benefits could be realized if the IT infrastructure and processes were aligned to support the efficiency and speed organizations desire.

This is where DevOps comes in. DevOps is an approach designed to more rapidly achieve transformation in both application development and IT infrastructure operations.

Organizations that succeed in achieving DevOps maturity are often characterized by their degree of mastery of six key capabilities:

- **Code, artifact, and binary management.** Repositories for retaining and managing software components
- **Configuration management.** Configuring and maintaining infrastructure and software systems in known ways
- **Cloud/PaaS.** Use of public, private, and hybrid cloud infrastructure to support software development
- **Containers.** Lightweight but highly scalable application runtime environments
- **Analytics.** Automated monitoring and management of infrastructure
- **Continuous integration/continuous deployment (CI/CD).** End-to-end automated processes that enable developers to write and automatically deploy code

One of these six areas in particular—containers—has taken on increasing significance for application development and infrastructure operations teams that seek additional speed and efficiency. This white paper addresses this move toward DevOps maturity with containers while highlighting one of the key issues that must be considered: managing persistent storage for stateful applications.

The Big Container Hurdle: Managing Persistent Storage for Stateful Applications

As DevOps teams begin to consider more real-world production deployments of containerized applications, challenges have emerged. Not the least of which is the management of persistent data storage for containers.

Do you need a better understanding of concepts such as data persistence with containers or stateful vs. stateless applications? See the brief inset on page 4.

Early in the application container movement, it became clear that the handling of data storage was a challenge. In a survey by the Cloud Native Computing Foundation (CNCF), nearly half the respondents (42%) said storage and resource management was a key container adoption challenge (Figure 1).[1] Many of these cited ongoing issues with storage persistence. Others wanted easier access to network storage.

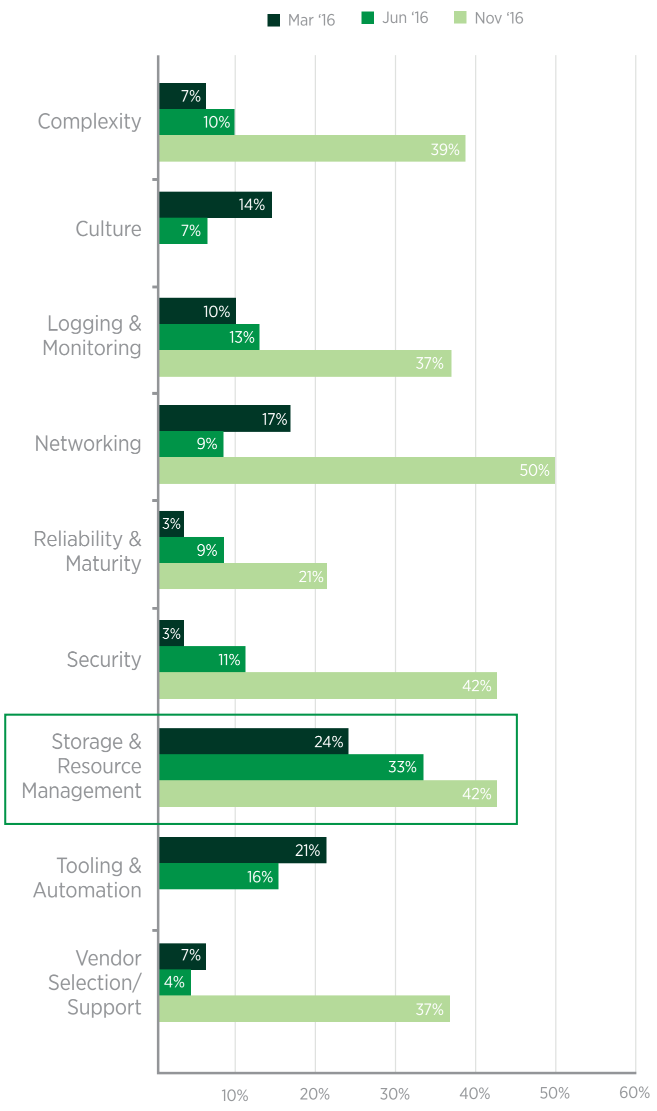


Figure 1) Challenges to container adoption (multiple responses were allowed).
Source: Cloud Native Computing Foundation.

As a result, container platforms began to tackle the persistent storage challenge. Early persistent storage provisioning efforts made a good start, but remained somewhat inflexible and overly manual. Accelerating the DevOps pipeline also did not seem possible, especially when applications needed to be coded, tested, and deployed into hundreds or thousands of containers.

From a scalability perspective, having to manually provision storage for thousands of containers began to look insupportable: too cumbersome, too error-prone, and too hard to maintain. There had to be a better way.

Before containers, many enterprise applications successfully met their persistent data storage needs through a central connection to shared enterprise storage. In a container world, couldn't such stateful applications operate just as well if they had easy access to similar shared storage features? But this question remained: How easy would it be to get there?

A Good Start: Basic Provisioning of Persistent Storage for Containers

Container environments such as Docker and Kubernetes originally responded to the storage persistence challenge with a useful, semiautomated mechanism. This mechanism allowed users to create and "claim" a particular persistent storage volume for use by one or more container processes. This required a storage administrator to first create persistent volumes from various underlying network storage resources.

Kubernetes: Basic Provisioning in Action
For Kubernetes, the process adopted code-based mechanisms associated with a PersistentVolume (PV) and a matching PersistentVolumeClaim (PVC).

A static, persistent volume (for instance, a volume with 8GB of all-flash storage capacity) would have to first be created by an administrator. Then the application user (or developer) could subsequently request or claim that specific persistent volume with a few lines of code.

This claiming and binding of PVs to PVCs was intriguing for container environments because it enabled containerized applications to start consuming persistent network storage through code.

Although a good start to enabling the consumption of network storage, a number of challenges with this basic provisioning approach remained. The largest of the challenges is the need for automation.

The Verdict: Some Progress, But Still Too Much Waiting

The use of preprovisioned PVs with their PVC pairings was an excellent first step to solve containers' persistent storage issues. However, much of the storage provisioning process still remained a largely manual, static, and repetitive effort on the part of both the Dev and Ops sides of the house. This posed several inefficiencies for application development and DevOps teams looking to reduce handoffs and automate their efforts toward CI/CD:

A Short Overview of Containers, Data Persistence, and Stateless vs. Stateful Apps

By their nature, containers are stateless. Their contents are ephemeral. This means a container's related applications or processes can be started, stopped, and restarted quickly during a given session. By default, this meant that data created while a container was active would be destroyed as soon as the container was terminated or destroyed.

Unfortunately, as DevOps teams looked to develop and deploy more applications into production on container platforms, a variety of needs emerged to retain or persist data past the life of any single container.

Many soon found that nearly every application had at least one process or microservice requiring a persistent volume (PV) for state data needing to persist past the life of a given container.

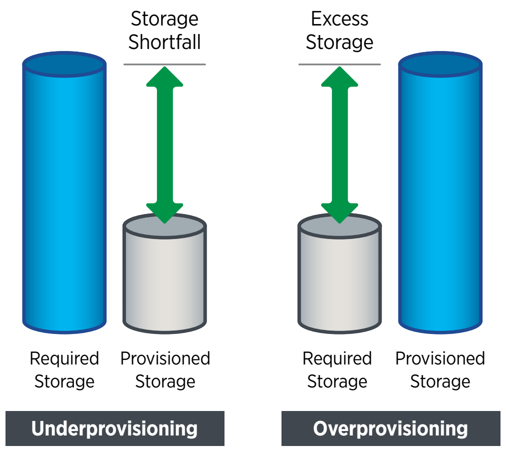
Here are a few examples of stateful applications requiring data persistence:

- **Database environments.** A database container needs to persist storage for its datastore. Given the ephemeral nature of containers, however, that's not feasible without help. Local storage was not a good option, either. If the container moved or was destroyed, it would lose access to that data.
- **Environmental or session data.** Stateful applications also often collect and save application environmental attributes or client session data (state). This acts as historical data to enhance the client experience. In this case, the next time the client interacts with the application, it can present relevant data or better manipulate data created in a previous session.

Just as important as the need to preserve state data was the need to share data. Organizations building or migrating to containerized applications soon found the need for everyone—from development to testing and operations—to access the same datasets from the same centralized network storage resources. Persistent volumes stored on network storage could also ensure that containerized applications were more protected by enterprise-grade storage features, driving greater application availability, reliability, security, and data protection.

- **Manual storage requests.** Application developers or QA personnel working on code or testing sprints might need to stop what they were doing to ask the storage administrator to create a PV. When scaling larger container applications, this might involve hundreds or thousands of ongoing storage requests.
- **Frustrating wait times for storage.** Development or QA teams might then be left waiting for approval of their request or for the administrator to schedule a time to manually create one or more PVs.
- **Inefficient use of storage.** With preprovisioned storage for PVs to support development and QA, the storage administrator could easily provision too little (underprovisioning) or too much (overprovisioning) storage capacity (Figure 2). They might also reserve too few or too many IOPS per volume:
  - **The risk of underprovisioning.** When too little storage is provisioned ahead of time for PVs, it could cause a bottleneck that limits developers requiring more capacity than what is readily available.
  - **The risk of overprovisioning.** Overprovisioned storage resources, now locked in one or more PVs, might later go to waste once bound to a PVC with a much smaller requirement for capacity or performance. (Remember those overprovisioned, underutilized storage silos of old?)

- **Ineffective use of administrator resources.** For multiple application development efforts, those involved in storage or cluster administration might find themselves the new bottleneck in one or more DevOps pipelines. If multiple new volume requests came at the final hour, the admin might still need to perform several manual steps in the setup and creation of each static PV. This practice also didn't aid administrators charged with further automating the infrastructure and monitoring its overall consumption.



(Figure 2) The risk of underprovisioning or overprovisioning.

If this sounds a lot like storage provisioning of the old days, you're not alone in thinking this way. In concluding his 2017 Tech Field Day talk, DevOps Through Desired State, NetApp Technical Marketing Engineer Andrew Sullivan commiserated with DevOps teams wanting to break free from this type of legacy provisioning. **"Storage is not a resource I should just have to tolerate, sulking down to the storage team's desk and begging for more capacity,"** he said. **"I don't want to provision storage. I don't want to consume storage in 2017 the same way that I did in 1989."** [2]

Thankfully, NetApp—and others who support container environments—knew there had to be a better way for DevOps teams to consume storage on demand and provision it dynamically exactly where and when it was needed.

Storage Classes, Storage Pools, and Trident: Persistent Storage on Demand

What Andrew Sullivan alluded to in the last section was the need for a better way to provision storage resources. This was a way that reflects the early promise of containers: Features dynamically created and deployed, on demand—with little to no manual intervention.

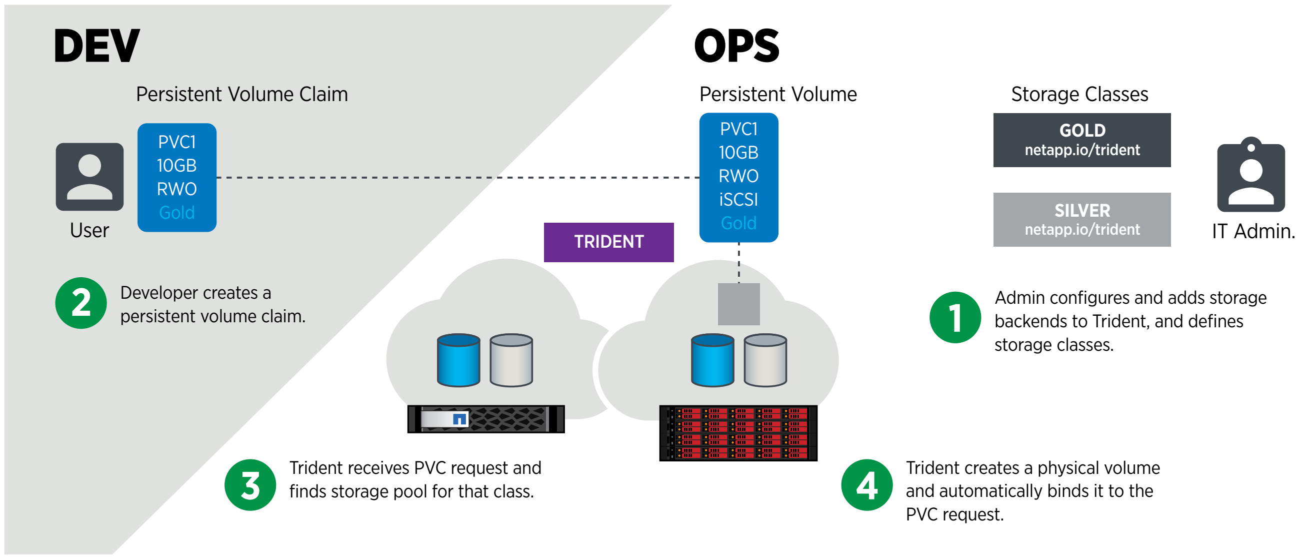
For Kubernetes, this meant the introduction of dynamic storage provisioning through another concept, StorageClass. It also meant some great automation from storage provisioners such as Trident for Kubernetes:

- **On-demand persistent storage.** The open-source Trident project, developed by NetApp, is a dynamic storage provisioner that gives containerized applications on-demand access to the persistent volumes of NetApp® storage they need, when they need them. No more waiting.
- **Unlocking powerful underlying storage**. Container environments using Trident are able to unlock powerful storage features from underlying NetApp data management platforms (such as NetApp HCI, ONTAP®, NetApp SolidFire® Element® OS, SANtricity®).
- **Not just for Kubernetes.** Similar functionality for persistent volumes is also available through Trident for Docker and OpenShift.

Could a Storage-Class Catalog Be That Easy?

Using Trident, developers in Kubernetes environments can now dynamically provision persistent volumes just by requesting a storage class from a virtual pool of underlying storage.

Let's take a look at how this works. Using storage classes, persistent volume provisioning occurs automatically on demand, using code. The user just makes a persistent volume claim and names a specific Trident storage class along with it such as Gold, Silver, or Bronze.


DEV
Persistent Volume Claim
User
PVC1
10GB
RWO
Gold
2 Developer creates a persistent volume claim.
TRIDENT
3 Trident receives PVC request and finds storage pool for that class.
OPS
Persistent Volume
PVC1
10GB
RWO
iSCSI
Gold
Storage Classes
GOLD
netapp.io/trident
SILVER
netapp.io/trident
IT Admin.
1 Admin configures and adds storage backends to Trident, and defines storage classes.
4 Trident creates a physical volume and automatically binds it to the PVC request.

(Figure 3) Dynamically provision persistent volumes using Trident.

Note: Configuration of underlying storage class attributes and naming conventions is a back-end function typically performed by an administrator when building an initial catalog of persistent storage classes. One organization's storage classes might just as easily be called Dev, Staging, and Production. Another's could be called Fast or Slow. For further details about these types of configuration functions, see Trident documentation.[3]

A persistent volume is then created from that storage class in the underlying NetApp storage pool. It is then subsequently bound to the user's persistent volume claim (PVC). Users no longer need to know about the underlying storage. Trident handles those details. That's it.

See Trident in Action
The best way to understand the value of Trident is to see its dynamic provisioning at work. Take a few moments to watch one of the following online demonstrations:

- **Short demo** (3:20 minutes): Using Trident for Dynamic Storage Provisioning with OpenShift [4]
- **Longer demo** (23:53 minutes): Managing Persistent Data in Kubernetes [5]

What does this type of provisioning functionality mean for different members of IT and DevOps?

- **For development or QA teams.** No more waiting for service tickets and storage request approvals. No more handoffs. Consumption of storage with a promised SLA that's just there when and where you need it, dynamically provisioned using familiar code interfaces. Developers now have a dynamic and flexible automatic provisioning system that frees them while still giving control to operations.
- **For IT or storage administrators.** No more scrambling for last-minute storage provisioning requests and endless service tickets. Less storage administration. More automated infrastructure scaling. More predictable control over storage consumption and easier monitoring of resources.
- **For IT executives.** Faster product delivery, better processes, and significant resource savings.

The Future of DevOps
DARZ, a full IT service provider, delivers DevOps agility with its Docker & Container-as-a-Service offering. This offering was based on NetApp all-flash storage and Trident for Docker. Customers can rapidly spin up or spin down application containers without the need for a full-fledged operating system, cutting compute requirements by as much as four times.

With a lean, flexible, containerized environment, customers can shorten test cycles, accelerate development, and deploy new products faster. By simplifying the container-storage interaction through the Docker volume set of commands, Trident makes managing persistent data in a Docker environment easy.[6]

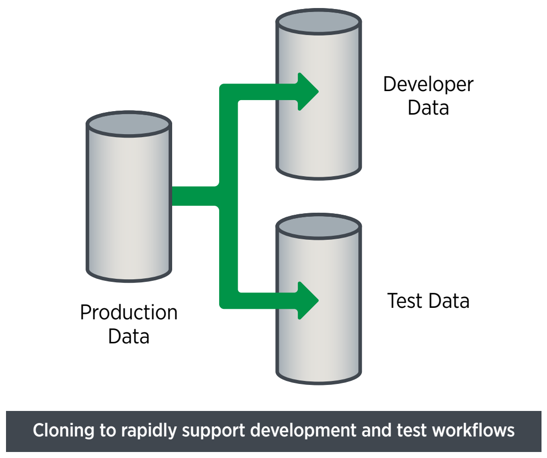
Transforming How You Work

Beyond its ability to dynamically provision persistent volumes, tools such as Trident also allow DevOps teams to do so much more.

This includes on-demand access, through code, to NetApp storage efficiency features such as Snapshot™ copies and cloning. Such features can be game changers for development or QA teams who need to do more in less time with the least amount of resources.

The use of Snapshot copies and cloning through NetApp and Trident allows:

- **Rapid creation of real-time clones of full production datasets.** This means new developer or test workspaces can be created dynamically in seconds with just a few lines of code. See Figure 4.
- **Quick and easy recovery of data.** With Snapshot copies, developers can quickly roll back datasets to a previous version. This is handy for testing code; developers can quickly iterate without fear of needing to recreate test datasets.
- **Storage capacity savings from Snapshot copies or clones.** Organizations that make multiple clones of their production data for development or testing often experience from 40% to 90%[7] storage capacity savings.



(Figure 4) The power of NetApp cloning to accelerate DevOps workflows.

Learn more about how NetApp cloning and Snapshot work with Trident:

- Self-service volume cloning with Kubernetes[8]
- Snapshot copies and self-service volume recovery with Trident[9]
- This Tech ONTAP podcast describes how NetApp Oracle expert Jeff Steiner cloned a large Oracle database in just 22 seconds using Docker with Trident[10]

Helping You, Helping Your Business

Solving problems for the needs of developers and engineers is not new to NetApp. In fact, this is our heritage. As a company, we learned early how storage infrastructure could be used to enable and accelerate the important work of developers, engineers, and the wider goals of their companies.
At NetApp, building better ways for organizations to manage data and consume storage is what we do. We see open ecosystems such as containers as a burgeoning, new area for further innovation. This innovation is propelled by its community members. NetApp is a part of this community and actively engaged in furthering its innovation. Toward that goal, NetApp is working on many ways to allow community members to more easily consume storage where and when they need it. Today, NetApp is proud to support seamless storage consumption across a variety of open ecosystems. We continue to develop one of the industry's most comprehensive sets of APIs and integrations for environments such as Docker, Kubernetes, OpenShift, OpenStack, Ansible, Chef, Puppet, and more.

Trident is but one example of these efforts. We encourage you to try Trident with your organization's container environments and look forward to hearing of the amazing efficiencies and savings it brings to your own DevOps pipeline.

Additional Resources

To learn more about Trident and any other NetApp DevOps integrations, we encourage you to check out the following resources:

More About NetApp and Trident

NetApp solutions for containers

Trident feature overview:
Introducing Trident

Introduction to Kubernetes persistent storage with Trident

Cloning: Introducing Volume Cloning with Trident for Kubernetes

Trident documentation

Trident GitHub download

More About NetApp with DevOps

NetApp solutions for DevOps

thePub (netapp.io)

NetApp Slack channel
(netapp.io/slack)

@NetAppPub

Endnotes

[1] "Meeting Challenges in Using and Deploying Containers," by Sarah Conway, April 27, 2017, Cloud Native Computing Foundation, https://www.cncf.io/blog/2017/04/27/meeting-challenges-using-deploying-containers/. Reproduced with red circle added, under Creative Commons CC-BY 4.0 license.

[2] "DevOps Through Desired State," Presented by Andrew Sullivan, NetApp, Day 14, May 11, 2017, Tech Field Day, https://www.youtube.com/watch?v=btLZI7M6gnY&list=PLinuRwpnsHacYmunO7zyES6SyrsrFfu5O&index=4.

[3] The latest Trident documentation can be found at https://netapp-trident.readthedocs.io/.

[4] "Using Trident for Dynamic Storage Provisioning with OpenShift," Online Demo, 3:20 min., by The Pub @ NetApp, Feb. 17, 2017, https://www.youtube.com/watch?v=97VZWYssL2E.

[5] "Managing Persistent Data in Kubernetes," Online Demo, 23:53 min., by The Pub @ NetApp, May 15, 2017, https://www.youtube.com/watch?v=XIuN91vG2wM.

[6] "DARZ Docker & Container-as-a-Service Drives Digital Transformation Through DevOps," Customer Success Story, NetApp, 2017, https://www.netapp.com/us/media/cs-darz-devops.pdf.

[7] "How NetApp IT Shortened Development Cycles Using FlexClone," NetApp Community Blog, by Gopal Parthasarathy, Oct. 8, 2015, https://community.netapp.com/t5/Technology/How-NetApp-IT-Shortened-Development-Cycles-Using-FlexClone/ba-p/110581.

[8] "Trident 18.01 beta 1: Introducing volume cloning to Kubernetes!" by Garrett Mueller, NetApp, Dec. 14, 2017, https://netapp.io/2017/12/14/trident-18-01-beta-1-introducing-volume-cloning-kubernetes/. (See also: "Trident 18.01 is Here," by Andrew Sullivan, NetApp, January 25, 2018, https://netapp.io/2018/01/25/trident-18-01/).

[9] "Self-Service Data Recovery using Trident and NFS," by Andrew Sullivan, April 3, 2018, NetApp,https://netapp.io/2018/04/03/self-service-data-recovery-using-trident-nfs/.

[10] "Episode 99 - Databases as a Service: Containers," Tech ONTAP Podcast, 2017, https://soundcloud.com/techontap_podcast/episode-99-databases-as-a-service-containers.

WP-7270-071