WHITE PAPER

A MODERN CI/CD PIPELINE ON PURE

CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY (CI/CD) ON PURE STORAGE

PURESTORAGE®

TABLE OF CONTENTS

INTRODUCTION

Traditional software development has evolved from a waterfall model to a more agile process that follows DevOps principles. In the traditional approach, writing code, compiling, testing packaging, QA, and release into production were done in silos. However, there is no adequate automation to test and identify for bugs quickly, which impacts software quality. This leads to higher technical debt and longer lead times for application deployment due to infrastructure provisioning. Disjointed and prolonged testing is inconsistent and delays the software-release process.

Continuous Integration and Continuous Delivery (CI/CD) is one of the most important and desired processes for modern code development and delivery for new and transforming software development environments. CI/CD provides shorter compilation or build times and faster release cycles with more automation and testing. Adopting CI/CD processes can drive innovation and faster time to market for commercial and enterprise applications.

CONTINUOUS EVERYTHING

The continuous workflow in the DevOps process is an automated and iterative process from development to delivery in the software development life cycle (SDLC). Various development, delivery, and platform tools and processes are used during workflow automation. The continuous process shortens the lead time for application development and deployment while accelerating the release cadence.

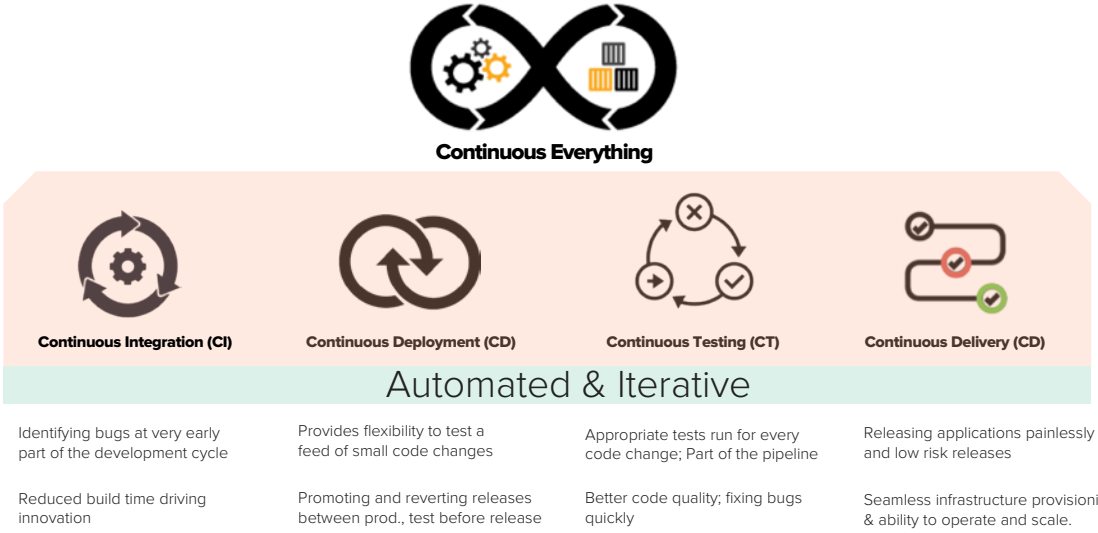

FIGURE 1. Continuous process from development to delivery

As illustrated in Figure 1, the following phases are the major parts of the CI/CD pipeline.

- **Continuous Integration (CI)** – During this process, developers identify bugs at early stages of the development cycle, fix them, and test in an iterative manner. Every time there is a new code change or a bug fix, the build or code compilation process takes place in the developer's private workspace. The developer then integrates the changes into the main code base. Depending on the size of the development team, these multiple builds could be running in parallel. Shorter build times lead to developer creativity that can breed innovation.
- **Continuous Deployment (CD)** – After the build process and packaging in the CI phase, the final build package is automatically deployed for user acceptance testing before it is released to production. More and more modern applications are running as microservices and containers as the unit of deployment on a platform. The platforms could be any public-cloud environment or containers for platform-as-a-service like Red Hat OpenShift or Pivotal Cloud Foundry. As containers are portable, *platform* is synonymous with standard operating system like Red Hat, Ubuntu, and others, abstracting the cores and memory required at application run time.
- **Continuous Testing (CT)** – Testing has recently started moving closer to the developers, in a move also known as "Shift Left." Using more automation to run various different tests appropriate to code changes is now part of the pipeline. This means that one simple change does not have to run the entire test suite. Instead, it focuses only on the relevant tests required for the change. Iterative testing and higher degrees of test success warrant much better code quality.
- **Continuous Delivery (CD)** – Continuous delivery differs slightly from continuous deployment. Both are continuous and automated, but the application is not released into production immediately with CD. There is a gate-keeping process where the latest build package goes through a test-automation process to validate compatibility and interoperability before it's released to production. Automating the process of promoting a new build into production and reverting back to an old build version is less painful and lowers risk.

CHALLENGES WITH THE TRADITIONAL CI/CD PROCESS

Data is generated during the entire CI/CD process. This data growth depends on the size of the development team(s), code base, number of builds/day, and deployments. The platform abstracts the infrastructure to be consumed as code. Data has to be stored, protected, managed, and reused during the entire SDLC. But data has gravity. Most of the phases in the CI/CD process are designed to function in silos.

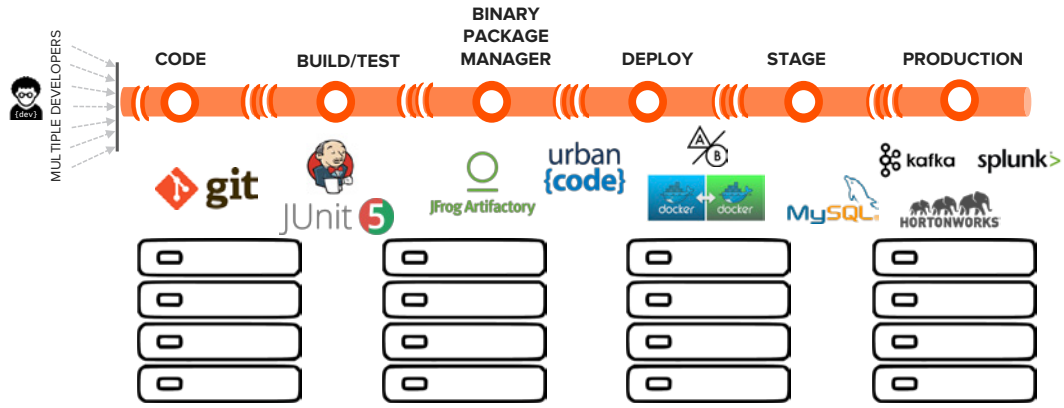

FIGURE 2. Traditional CI/CD application development process

A typical SDLC pipeline consists of six stages – plan/code, build/test, packaging, deploy, stage, and release into production.

Step 1: Code – Developers perform "git clone and git push" operations to and from their workspaces in their laptops to write and update code. There is a performance impact on the Git server, where there can be hundreds and thousands of developers checking out code simultaneously. There is also a security concern about having the proprietary code moving around on individual user laptops outside the business premises.

Step 2: Build/Test – While making code changes and updates in the workspaces, developers are supposed to run private builds and tests before merging the code to main or development branches. Once the respective branch is updated with the code changes, integrated builds are triggered along a prescribed set of CI tests to check the validity of the code. These builds and tests run in parallel, which requires higher throughput and concurrency with lower job failure objectives.

- Scalability and concurrency are a limitation with local storage.
- Service-level objective: bandwidth, IOPs, and latency are not properly met.

Step 3: Binary Packet Manager – Successful builds are then packaged with the right dependencies and promoted to the next CD phase. You must replicate or move the data from the repository that has all the binary packages to stage it, then release it into production. As the size of the business grows, many versions and new package types are generated and stored. Additional servers to support the large dataset size results in server sprawl with high manageability overhead.

Step 4: Deploy and Stage – During the CD process, applications are tested with corresponding databases to perform functional tests for new features. It's not trivial to get a copy of the production database or set up a new one for testing purposes. It is time-consuming and involves more risk to get a copy of the production database. Obfuscation of certain parts of the data in the database limits developer innovation.

Step 5: Production – The following questions arise when the application is ready to be released into production:

- How do I provision the underlying infrastructure on-premises or move to a public cloud to release it?
- How challenging will it be to operate, manage, and monitor the application and the corresponding data ingestion?
- Is public cloud a viable platform to store high data growth and scale performance with respect to cost?

PURE STORAGE® FOR CI/CD PIPELINES

The term "storage" may sound like an aberration to developers and other users like data scientists. However, data is important. The developers and end users always prefer to have access, control, continuity, and availability of data. However, using a data platform like Pure Storage DirectFlash™ helps, directly and indirectly, to accelerate and optimize the CI/CD pipeline. Pure Storage also provides a high return on investment (ROI) using Pure Evergreen Storage™ Subscriptions (ES2) with a standard data hub for data pipelines like SDLC, analytics, AI, and many more workflows.

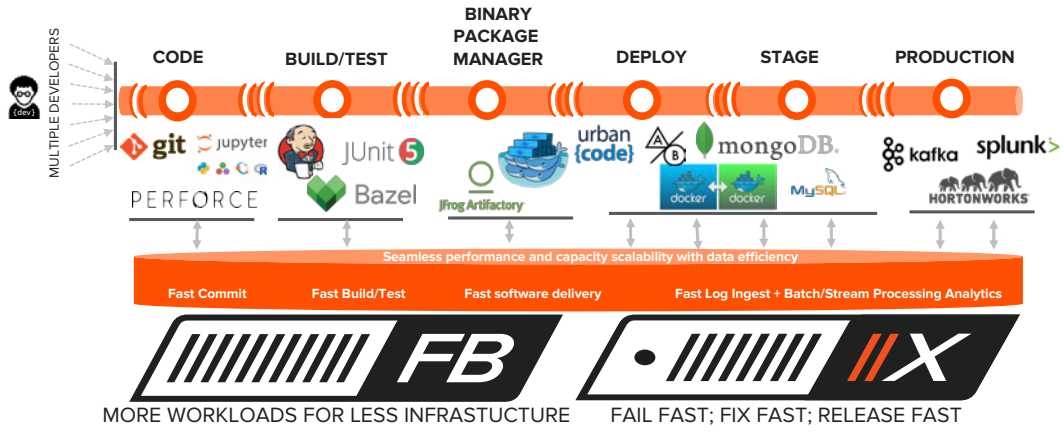

FIGURE 3. Modern CI/CD pipeline

Pure Storage has a product portfolio that has been growing ever since the company's inception. For the purpose of validating a reference architecture for the CI/CD pipeline, Pure's FlashBlade™ and FlashArray™ products are used in the CI and CD processes respectively.

- FlashBlade is a highly-scalable and distributed file-system and object-based data platform that supports NFSv3/v4.1, SMB, HTTP, and S3 protocols. This file-based platform helps to provide high concurrency and parallelism during the CI process for shortened job completion times, along with data compression of 2:1 and above. You can use the same platform to rapidly deploy new applications that can be used for analytics and AI workflows. You can provision more workloads on a standard data platform like FlashBlade that can scale capacity and performance.

- FlashArray is a block-based data platform that supports iSCSI and Fibre Channel protocols. The block-based platform provides rapid provisioning of replicas for the new application version during the CD process without downtime. Developers can test applications with databases and recover from failures quickly before releasing them to production with data reduction of 10:1 or more. The application can also be ported to Pure Cloud Block Store (CBS)[1] in AWS.

CI/CD REFERENCE ARCHITECTURE ON PURE STORAGE

The use of containers and virtual machines (VMs) is very common in most of the modern CI/CD process. The proposed reference architecture in this paper uses containers for the CI process and VMs for the CD process on FlashBlade and FlashArray respectively. However, you can implement the CI/CD workflow in many different scenarios – it is not one size fits all.

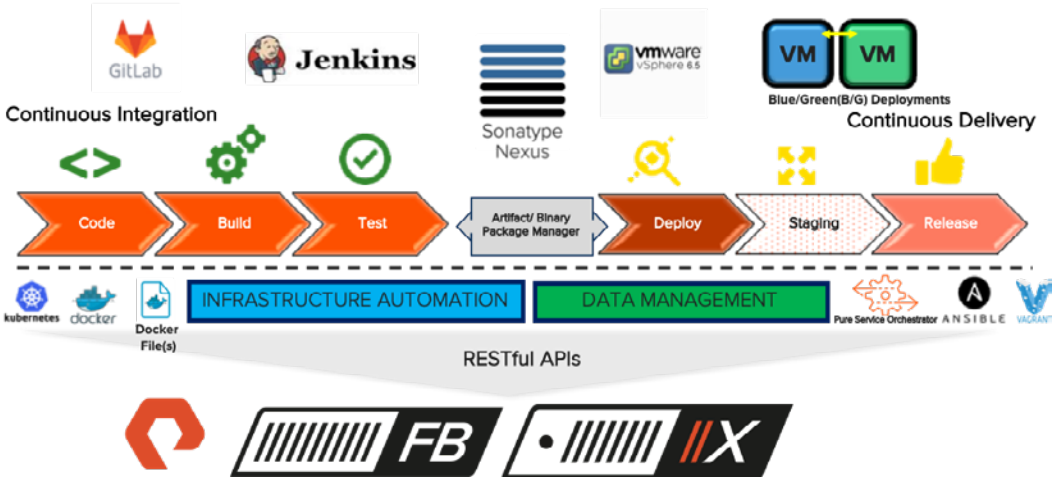

FIGURE 4. CI/CD workflow validation on Pure Storage products

The spirit of CI/CD is to be an entirely automated and iterative process that can use conventional or modern development and automation tools designed to be more synonymous to business requirements. While the tools may vary in the CI/CD pipeline, having a standard data platform for various different workloads not only accelerates the workflow but minimizes the dependencies of storing, moving, and managing data in on-premises, private cloud, public, or hybrid-cloud environments.

CONTINUOUS INTEGRATION WORKFLOW

The CI process in this reference architecture uses Kubernetes cluster and Pure Service Orchestrator™ (PSO) to provide persistent storage on demand from FlashBlade for persistent volume claims (PVCs). The PVCs are created by different tools like GitLab, Jenkins, and Nexus that are running in the Kubernetes cluster during the CI process.

[1] Currently in beta

- **Gitlab** – Gitlab source code management (SCM), also known as a version-control system, manages the different code changes in the source code repository. Git is one of the most common, popular open source and distributed source code management tools. Gitlab is one of several flavors of Git. The components of Gitlab are set up in the Kubernetes cluster using helm chart, which uses FlashBlade over NFS for scalability and high availability.
- **Jenkins** – Jenkins is a commonly used open source CI tool. The Jenkins pipeline is automated to perform all the CI and CD phases in this architecture. Jenkins allows developers to build and test code to identify and resolve bugs quickly, in an automated manner. Jenkins follows a distributed architecture with a primary/secondary configuration. The Jenkins "primary" is responsible for scheduling, managing, dispatching, and monitoring build jobs. Each of these jobs represents a "secondary." The secondaries run different jobs as requested by the primary in a distributed manner. It is recommended to set up the Jenkins configuration directory over NFS on FlashBlade for high availability and resiliency. When a Jenkins secondary dies, it can spin up in another pod in the Kubernetes cluster pointing to the NFS share on FlashBlade that has all the configuration files.
- **Nexus** – Nexus is an open-source, binary package manager tool that supports different repositories like Docker images, NPM, Python, etc., and their dependencies. The dependencies are required during the private and integrate builds during the CI process. The final build package is pushed into Nexus that promoted to the CD phase to deployed, staged, and released in to production. The docker images for NPM dependencies are required to compile the source code in the CI process. Another for the Vagrant files is used in the CD process for this reference architecture validation.

SCOPE OF VALIDATING THE REFERENCE ARCHITECTURE

For the purpose of this reference architecture, we've used "wordpress" source code to build a new version that changes the original "wordpress" image and replaces it with the Pure Storage logo when the updated application is released into production.

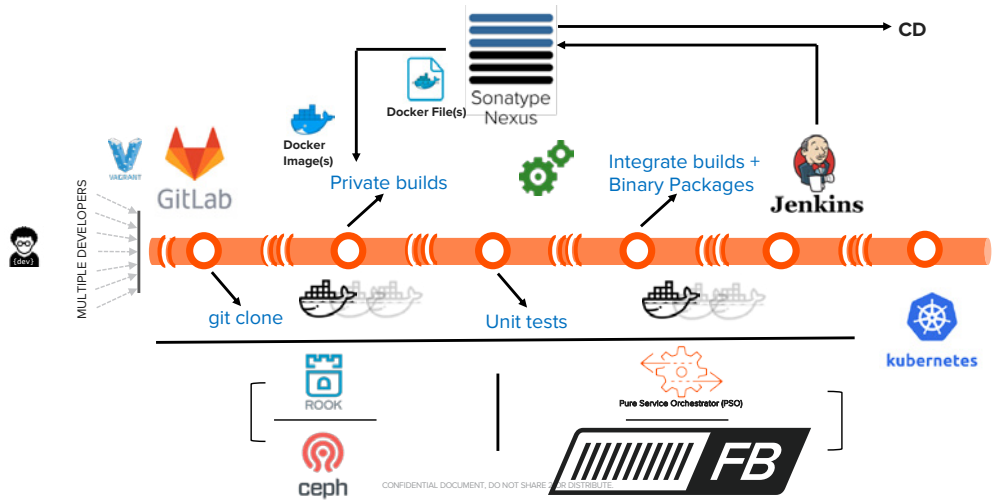

FIGURE 5. CI process compared with local SSD vs. FlashBlade

The following scenarios were validated in the CI process, as illustrated in Figure 5:

1. Set up a CI pipeline with Gitlab, Jenkins, and Nexus running on local SSD storage. Use Rook to dynamically provision block storage using Ceph.
2. A similar pipeline was set up on FlashBlade using PSO.
3. “Wordpress” source code was downloaded to Gitlab.
4. A docker image was created with the “npm” dependencies (grunt and bower) that will be required for compiling the “wordpress” source code. This image is stored in the docker registry in Nexus.
5. The pipeline performs the following operations:
   - Applies Layer 4 load balancer and Layer 7 ingress using MetalLb and Nginx respectively for tools like Gitlab, Jenkins, and Nexus. MetalLB was deployed on the cluster to scale the “git clone” operations, do TCP communication between the Jenkins primary and secondary(ies), and push the new binary package(s) to Nexus. The Nginx-based ingress helped to route public requests to various services such as Jenkins, GitLab, and Nexus from their respective web UIs.
   - Creates user workspaces for every developer. Every workspace pod in the Kubernetes cluster uses the above docker image and has persistent storage provisioned for itself. The persistent storage is either provisioned on local SSD storage using Rook and Ceph or on FlashBlade using PSO for performance validation (mentioned later in this section).
   - Pulls the source code using “git clone” and other required dependencies.
   - Performs a private build in the user workspace.
   - Makes necessary changes, like replacing the main Wordpress picture with the Pure Storage logo and including comments along with the blog posts. The pipeline runs automatic unit tests on these changes.
   - Pushes the code change (git push) to the Gitlab repository. Automatically kicks off the integrate build in the Jenkins pipeline.
6. The build is finally packaged with the required dependencies and pushed into Nexus in a .gz file.

The scalability test is applied to the following scenarios:

- The number of jobs in the pipeline is called from 50-100-200.
- While scaling with the number of jobs, the “git clone,” “private builds,” “unit tests,” and “build packages” job completion time(s) were measured for both scenarios – local SSD and FlashBlade.

TEST RESULTS

Test results indicated that the local SSD storage performed very well with a smaller number of jobs. However, as the number of jobs scaled from 50 to 100 to 200, FlashBlade performed much better with almost no job failures. Conversely, the local SSD storage demonstrated about 20% job failures with 200 concurrent jobs.

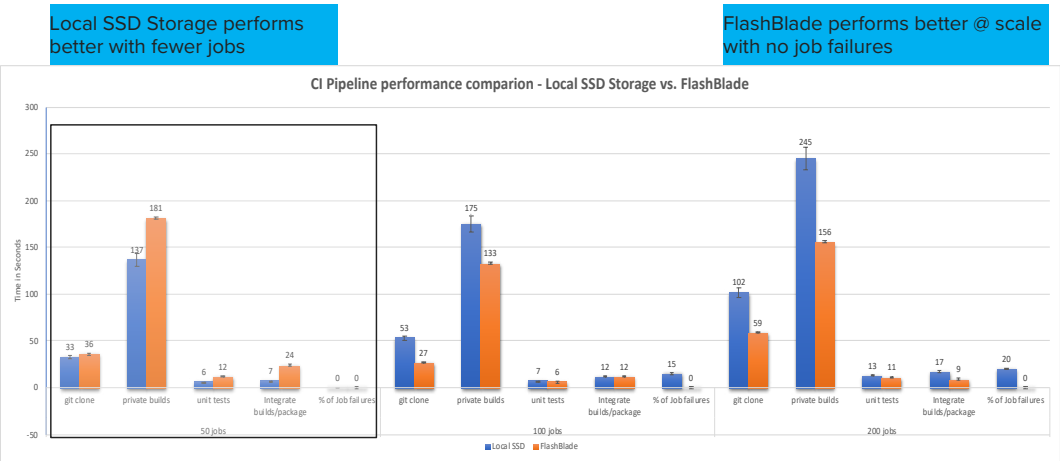

FIGURE 6. Performance comparison with 50, 100, and 200 concurrent jobs

The tests indicate speed improvements over SSD:

- Build jobs at scale finished 25% faster than local SSD storage as the jobs scaled to 200.
- The “git clone” operation to create 200 workspaces and sync the source code locally was 45% faster than local SSD storage.
- The unit tests and build package contributed to a 13% improvement over local SSD storage at scale.

OBSERVATIONS

Application portability, modularity, and tool integrations can all happen in the CI workflow layer to make the process automated and iterative. It’s undeniable that the platform and underlying data platform like FlashBlade do provide a significant speed boost to complete jobs in the pipeline.

The fact that the local SSD storage could not perform beyond a small number of jobs indicates that FlashBlade can handle a high number of concurrent requests in parallel and at scale. With an average of 27% improvement in the entire CI process and a space savings of up to 2:1 in this validation, FlashBlade provides higher ROI and lower total cost of ownership in the application development phase.

Using FlashBlade as a standard data platform removes the ambiguity of server manageability and data silos. FlashBlade functions as a data hub by consolidating data silos and providing predictable performance to the various different workloads in the CI process. The CI pipeline on FlashBlade provides a value stream that feeds into other data pipelines, like the CD process, AI, analytics, and others. As the number of containers scaled from 100 to 200, FlashBlade demonstrated a high degree of concurrency with multiple TCP sessions for iterative and continuous running jobs during the CI process.

When traffic was routed through L7 ingress (Nginx), the network I/O was impacted and was not able to scale properly. MetalLB as an L4 load balancer helped to scale all the services that were exposed for git clone, push build artifacts, etc., and was critical for Jenkins while communicating with the secondaries over TCP.

CONTINUOUS DELIVERY WORKFLOW

The CD pipeline may vary with different businesses and application types. After going through rigorous testing and the final build, you can deploy a web application and promote it into production as new updates are made to the web pages. There are many scenarios in which new application changes after every deployment require additional testing with a database before releasing into production. The latter is used to validate in the reference architecture.

Tools Like IBM UrbanCode and Cloud Private, Red Hat OpenShift, and Pivotal Cloud Foundry are used to deploy, stage, and release applications to production. The reference architecture in this paper does not use any of these tools. It uses an Open Source HAProxy tool to perform a blue-green deployment for testing and releasing the application to production.

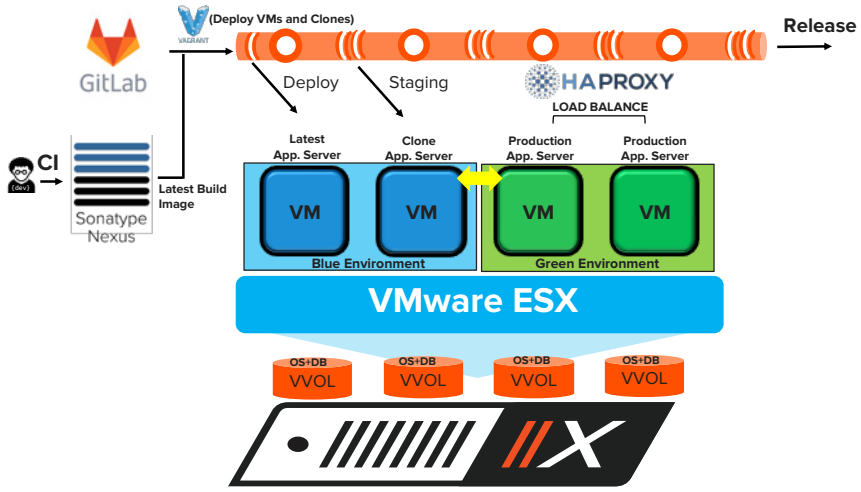

Figure 7. CD pipeline with blue-green deployment

The CD pipeline consists of two parts, as shown in Figure 7.

- Deploy the application from Nexus on a vSphere VM and clone the VMs for testing the database during the staging phase.
- Use an HAProxy to seamlessly switch the current production version of the application to the most recent version with the changes.

DEPLOY AND STAGE

One of the major challenges in the CD phase is provisioning a production-like database to validate the changes made to the application before releasing into production. In this reference architecture, we need to test the "wordpress" changes to include comments to each blog post. In order to test it, we set up the production MySQL database manually, using migration scripts. Spinning up multiple copies of the database to run multiple tests, or revert back if any test fails, is time-consuming, further delaying the release of the application to production.

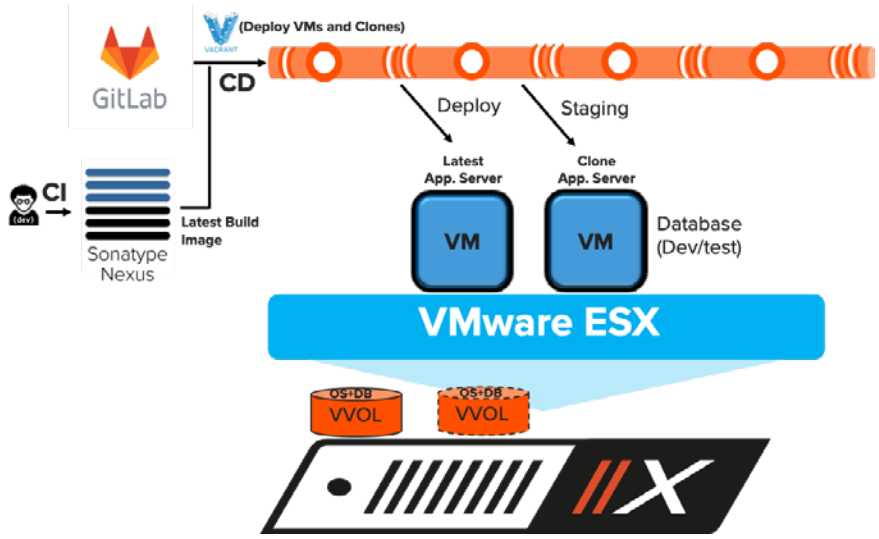

FIGURE 8. Continuous deployment for staging on FlashArray

In this reference architecture, Vagrant and a vSphere plugin are used to spin up a VM with an Ubuntu image. The final build package that the CI process pushed into Nexus is automatically installed on the new VM. The "wordpress" application used in this reference architecture installs the MySQL database on the same VMware Virtual Volume (VVOLs). All these operations are automated in the CD pipeline.

The VM creates a VVOL on FlashArray. VVOL datastores are array-aware and communicate with VSphere from FlashArray using VMware API for Storage Awareness (VASA). The VASA provider converts the VMware-issued API calls to FlashArray-specific requests. VVOLs is recommended over regular Virtual Machine Desk (VMDK), as it plays a significant role in cloning the production database instantly for testing the "wordpress" application without additional space on FlashArray.

The integrated solution with FlashArray and vSphere 6.5 in the CD pipeline automates the creation of instant production like MySQL databases to test the "wordpress" changes. You can create multiple clones of the VM to test the new application changes. In this reference architecture, we tested 1,000 blog posts with 1,000 comments for each blog in the "wordpress" application. Adding the "comment" to the application was a new feature. Once the test passed successfully, the original VM and its clones were ready to be promoted to production.

The first part of the CD phase alleviates the challenge of provisioning a production-like database quickly for testing the application. The developer would provision a production-like database to test new features, such as adding "comments" in this validation.

Instantly cloning the production database on FlashArray reduces the time for manually migrating the database. This mitigates the risk from the production database and makes the developer more productive. The VM clones on FlashArray did not take up additional space. When new records were inserted to the cloned database VM during the staging phase, a 24:1 data reduction was achievable.

HAPROXY AND PRODUCTION RELEASE

HAProxy is an open source tool that functions as a reverse proxy and TCP/HTTP load balancer. HAProxy consists of three main components: back end, front end, and access control list (as shown in Figure 9). For the purpose of this reference architecture, we used Layer 4 load balancing. The HAProxy pipeline automates the process of promoting back-end servers to production without any downtime using the reverse-proxy feature.

The old version of the application coexists as the new version is running in production. This allows you to revert to the old version in case the new production version seems unstable or has issues from with external client access. If the new version is stable, the demoted version will soon be replaced by a newer build and this cyclical process continues.

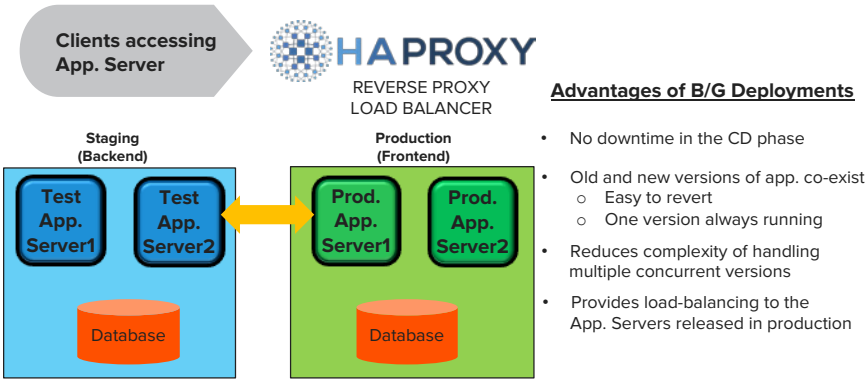

FIGURE 9. Blue-green deployment using HAProxy

In this reference architecture, the application server VMs and their cloned VVOLs represented in blue (Figure 9) are promoted to a green deployment and seamlessly into production. Now the production version of "wordpress" has the Pure Storage logo with the "comments" added to all blog posts. Get the sample scripts used for this reference architecture.

CONCLUSION

Modern CI/CD workflow requires speed along with automation and repeat value. While developers, data scientists, and other end users interface with the CI/CD workflow at a higher level, the process feeds off the underlying data platform to store, protect, access, and control data.

Pure Storage provides an integrated solution with RESTful APIs and Pure Service Orchestrator to provide a zero-storage touch and accelerate the CI/CD workflow from design to release. FlashBlade and FlashArray remove silos and provide a value stream with reduced software development lifecycle time and minimum manageability overhead in the overall CI/CD process. FlashBlade provides performance and capacity scalability. FlashArray removes ambiguities to provision databases for dev/test before releasing applications to production.

ABOUT THE AUTHOR

As a Technical Director for DevOps/EDA, Bikash Roy Choudhury is responsible for designing and architecting solutions to address customer business requirements in their transition to agile development and application workflows across industry verticals that include EDA/high tech, financial services, gaming, social media, and web-based development organizations. Bikash has also worked on validating solutions – with Red Hat OpenStack Platform (IaaS), Apprenda (PaaS), Kubernetes/Docker, Jenkins, JFrog Artifactory, Ansible, IBM Private Cloud (PaaS), and Perforce Helix – using RESTful APIs and integrating them with data platforms that provide persistent data storage in private, hybrid, and public clouds. With over 26 years of experience, Bikash has a deep understanding of workloads and workflows that have transitioned from spinning disks to flash over the years, via NFS/blocks and S3 object store, and of automation and data management strategies for end users and business owners.

ps_wp15p_modern-ci-cd-pipeline-on-pure_01