

A vertical strip on the left side of the page shows a portion of a blue architectural drawing. It includes technical lines, circles, and text such as "FLOOR R", "SPRINKLER", "TO DRAIN", and "TRANSITS".

THE VERISCALE ARCHITECTURE: ELASTICITY AND EFFICIENCY FOR PRIVATE CLOUDS

Mikael Lofstrand, Sun Microsystems

Sun BluePrints™ Online

Part No 821-0248-11
Revision 1.1, 09/22/09

Contents

Introduction.....	1
Requirements of a Scalable Datacenter.....	3
VeriScale Architecture Overview	4
VeriScale Automation	4
VeriScale Functional Components	4
Project OpenSolaris™ Dynamic Service Containers (OpenSolaris DSC)	7
Payload Lifecycle.....	8
An OpenSolaris DSC Use-Case	11
Service Delivery Network (SDN) Architecture and VeriScale Automation.....	12
Services.....	13
Management.....	13
Security.....	13
Network Optimization Through Distribution	15
Deployment Optimization.....	15
Load-Balancing Optimization	15
Load-Balancing on a Traditional Architecture	15
Load-Balancing Utilizing OpenSolaris DSC.....	16
Load-Balancing on a VeriScale Architecture.....	17
Network Data Path Optimization on a VeriScale Architecture	19
Benefits of VeriScale Optimization Through Distribution	19
Proof of Concept for the VeriScale Architecture	20
Objective.....	20
Platform.....	21
Software	21
Servers.....	21
Network	21
Setup	22
Execution	22
The VeriScale Concept Proved	23
The VeriScale-Enabled Business Process	24
Conclusion	25
About the Author	25
Acknowledgments	25
References.....	26
Ordering Sun Documents.....	26
Accessing Sun Documentation Online	26

Chapter 1

Introduction

The modern datacenter is evolving into the network centric datacenter model, which is applied to both public and private cloud computing. In this model, networking, platform, storage, and software infrastructure are provided as services that scale up or down on demand. The network centric model allows the datacenter to be viewed as a collection of automatically deployed and managed application services that utilize underlying virtualized services.

Providing sufficient elasticity and scalability for the rapidly growing needs of the datacenter requires these collections of automatically-managed services to scale efficiently and with essentially no limits, letting services adapt easily to changing requirements and workloads. Sun's VeriScale architecture provides the architectural platform that can deliver these capabilities.

Sun Microsystems has been developing open and modular infrastructure architectures for more than a decade. The features of these architectures, such as elasticity, are seen in current private and public cloud computing architectures, while the non-functional requirements, such as high availability and security, have always been a high priority for Sun.

The VeriScale architecture leverages experience and knowledge from many Sun customer engagements and provides an excellent foundation for cloud computing. The VeriScale architecture can be implemented as an overlay, creating a virtual infrastructure on a public cloud or it can be used to implement a private cloud.

This Sun BluePrints™ article provides an overview of Sun's VeriScale architecture. The article includes details of VeriScale's defining principals, underlying components, interactions, and advantages. Applicable real-world scenarios are also described.

The article addresses the following topics:

- “Requirements of a Scalable Datacenter” on page 3, describes the requirements that are addressed by the VeriScale architecture.
- “VeriScale Architecture Overview” on page 4, describes the automation layer and functional components of the VeriScale architecture.
- “Project OpenSolaris™ Dynamic Service Containers (OpenSolaris DSC)” on page 7, describes the components of OpenSolaris DSC that are used by the VeriScale architecture.
- “Service Delivery Network (SDN) Architecture and VeriScale Automation” on page 12, describes how the SDN architecture is used to implement the VeriScale automation requirements.

- “Network Optimization Through Distribution” on page 15, describes how the VeriScale architecture can be used to distribute and optimize application deployment and load-balancing.
- “Proof of Concept for the VeriScale Architecture” on page 20, describes a proof-of-concept (POC) that was implemented to verify the feasibility and utility of the VeriScale architecture.
- “The VeriScale-Enabled Business Process” on page 24, describes how the Veriscale architecture concepts can be applied to business processes in a software intensive enterprise.

This Sun BluePrints article is targeted at technical IT managers and datacenter system architects who are responsible for implementing, managing, or recommending cloud computing environments and implementation strategies. The article assumes that the reader has a good understanding of network centric datacenter architectures and modern networking concepts.

Chapter 2

Requirements of a Scalable Datacenter

Evolving modern datacenters to support cloud computing requires infrastructure components and architectures that can grow and contract on demand, and permit self-deploying, self-provisioning, and elastic services. In this model, the datacenter is viewed as a collection of managed application services that rely on the underlying infrastructure services — processing, networking, and storage — along with their management services.

Additionally, the requirements of a network centric datacenter architecture include:

- Providing a simple mechanism to easily deploy components so that the operational IT environment is efficient
- Supporting a rapid development cycle to service new requirements and deploy new services
- Avoiding reliance on centralized provisioning systems that inherently have breaking points that limit scalability
- Consisting of self-contained components that include a fully-operational, self-sufficient software stack with applications, virtualization technologies, storage, and networking for modularity and ease of deployment
- Enabling the self-provisioning of services, while providing cost-efficient, scalable, and rapid deployment with standard technologies
- Enabling load-balanced services, so that the quality of service provided to clients is not dependent on the specific instance of the service that the client accesses

The VeriScale architecture is applicable to public or private cloud computing datacenters as well as other network centric datacenters — with an inherent focus on scalability and efficient deployment of replicable services.

Chapter 3

VeriScale Architecture Overview

The objective of the VeriScale architecture is to support the dynamic infrastructure required by dynamic, network centric datacenters by automatically managing replicable scaled-down prototypes. The prototypes are stored as payloads of self-contained, packaged software that are deployed to servers.

VeriScale Automation

The VeriScale units of management are services, not individual functional components. Networking functions are embedded within services so that services are fully functional, self-contained, self-sufficient, and self-provisioned networked entities that can be viewed as replicable prototypes of the service they provide.

The traditional approach to automation is to implement centralized provisioning systems that deploy individual services onto target systems. This approach limits scalability since regardless of the size of the centralized system, there is always a limit to its capacity and throughput. The centralized system can break if there is a demand peak for its services that outstrips its capacity. In contrast, the VeriScale architecture assigns the deployment logic to the target systems, creating an environment where the targets are self-updating — pulling components from the appropriate repositories as required. This strategy improves scalability as the provisioning workload is handled by each target system and not by a centralized resource. Contrast this approach with the very limited level of network automation in traditional architectures and the true power of VeriScale is immediately apparent.

VeriScale Functional Components

Sun's VeriScale architecture (Figure 1) is comprised of an integrated set of functional components described in the following sections.

- **Point of Delivery (POD)**

The point of delivery (POD) is part of Sun's scalable dynamic infrastructure suite. The dynamic infrastructure suite represents a self-contained network service stack that includes both software and infrastructure devices. Each POD provides a certain defined functional capacity and can range in size from a single hardware thread on a multithreaded server to any number of server racks in a data center. Each POD is delivered ready to be easily plugged into the network and participate in the system.

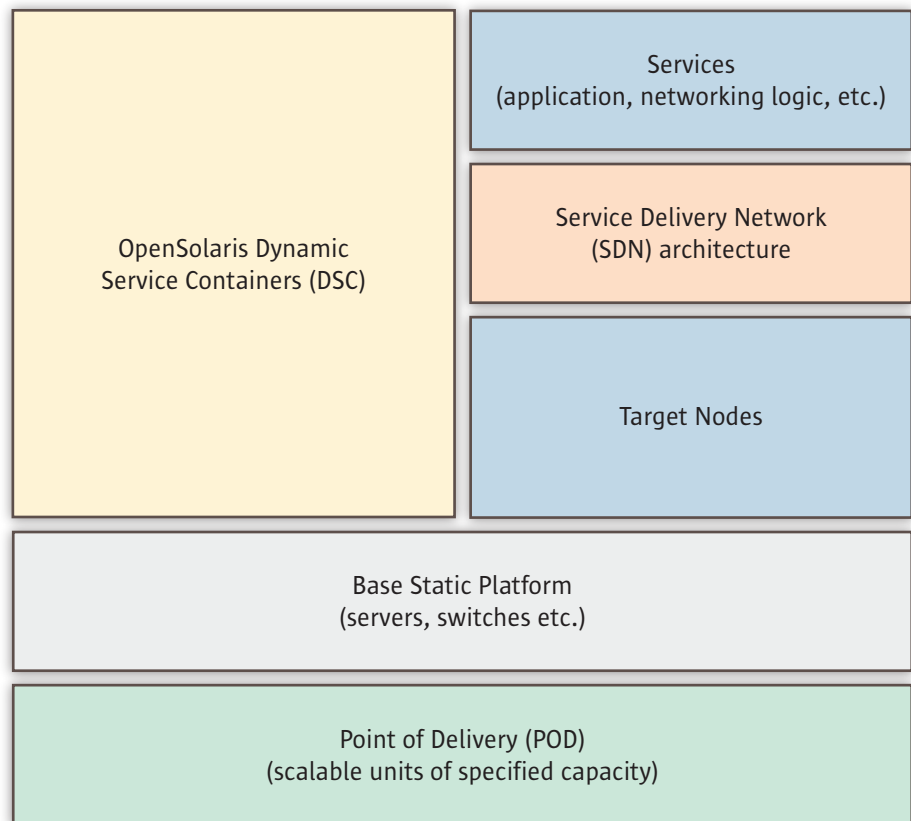


Figure 1: VeriScale architecture layers

- **Base Static Platform**

The base static platform (Figure 2) consists of the interconnected nodes and storage devices that are physically wired and remain statically connected. These infrastructure components are virtualized in higher layers that are dynamically reconfigured as needed. The components that are used for this layer are defined per required cost, performance, availability, interfaces, etc. The base static platform can be implemented as a flat network where virtual LANs (VLANs) can exist anywhere on the platform or as a POD.

- **Target Nodes**

The target nodes are the actual compute hosts that run the applications. The Sun service delivery network (SDN) architecture provides a set of network connectivity, routing, load-balancing, and security mechanisms that combine to form a flexible network infrastructure design framework. The SDN architecture provides high performance, scalability, availability, security, flexibility, and manageability for the datacenter infrastructure. The SDN methodology supports a common approach to designing network architectures, and provides a common set of tools that help ensure proper architectural design decisions and trade-offs.

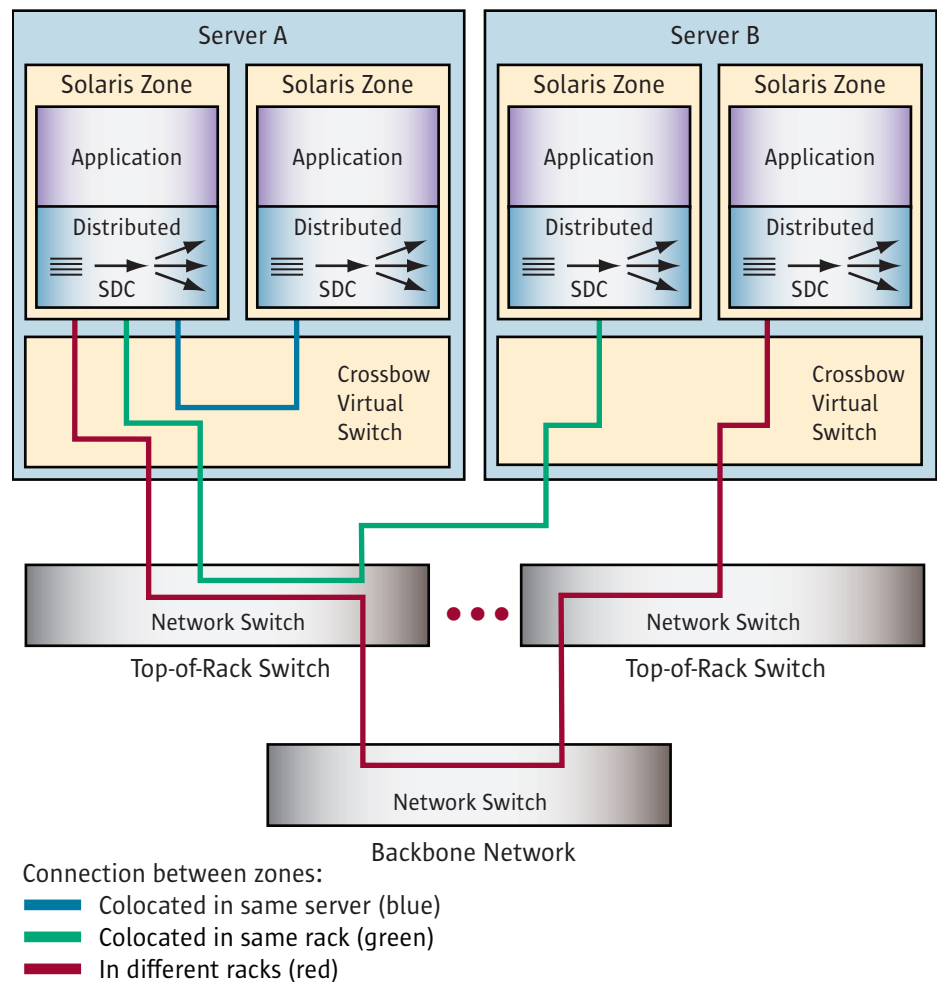


Figure 2: The base static platform

• Services

The services encapsulate the business functionality provisioned at the POD such as infrastructure services or applications. The services include their required networking logic. The functional components are managed and automatically deployed through OpenSolaris DSC.

Chapter 4

Project OpenSolaris™ Dynamic Service Containers (OpenSolaris DSC)

OpenSolaris DSC is a distributed resource and service management system designed for simplicity and scalability. It is suitable for lightweight applications with simple installation and configuration requirements. OpenSolaris DSC tracks what is going on in the network as a whole, maintains multiple hosts that run the OpenSolaris DSC services and software, and activates and deactivates payloads.

OpenSolaris DSC implements a pull model that uses resources on the target nodes for provisioning, thus avoiding the resource bottlenecks associated with a centralized provisioning system. In this model, the resources available for provisioning grow as the number of target nodes increases. The performance limitations then become associated with the target nodes and are not related to the number of target nodes, but rather to the load-capacity of each single target node. The components of OpenSolaris DSC are described in detail in the following sections.

- **Registry**

The registry holds the status, administrative, and operational data needed to deploy and manage services across the scalable architecture. This data includes capacity information for hardware, information that is used to deploy the correct number and configuration of service instances, and the data needed to download and install payloads.

- **Repository**

The repository is a simple, passive storage facility for the payloads that can be co-located with the registry on the same compute/storage device. The repository functions as a file server for the node controllers to download payloads.

- **Node Controllers**

Node controllers locate the registry and continuously query it for changes of desired state, pull initial configuration data, and query the registry for the desired state of service definitions. In addition, node controllers analyze the suitability of a given POD to host a required workload. When a service is provisioned, a node controller offers to host a workload, downloads and installs the needed payload, starts the service, and updates the registry with its state.

Node controllers can be customized for specific functions. One example is described in the section “Load-Balancing Utilizing OpenSolaris DSC” on page 16, where a node controller updates the configuration of a load balancer from the

registry. Another example is a node controller that updates the registry to increase the number of deployed service instances for a service whose response time has slowed to an unacceptable level.

- **Nodes**

Nodes are aggregates consisting of computing, storage, and communication devices. These aggregates have a predefined application capacity that simplifies the resource management model, and allows the dynamic reprovisioning of applications as their needs grow. Nodes are added to an already-provisioned application when its capacity is exhausted, or reassigned when they are no longer needed. Alternatively, the applications can be redeployed to larger or smaller environments as needed.

- **Payloads**

Payloads maintain information and data needed for the node controller to install and run a workload. Some examples of workloads include an install script, a binary, base configuration, and additional content. The payload can also include a load balancer and its initial configuration or an application running in an immutable service container¹ to enhance security for the running workloads.

Payload Lifecycle

Payloads are self contained, fully configured entities and as such they can consist of applications, application platforms, or entire virtual machines (VMs). For example, in the case of a payload that encapsulates a Web server, the payload includes the Web server software, some basic generic content, and enough configuration information to allow the Web server, once deployed, to obtain dynamic content from a virtual data store server. In addition, as described in the section on “Network Optimization Through Distribution” on page 15, payloads contain a virtual data store service that is represented by an internal load balancer. This uniformity helps enable identical configurations for all of the Web servers, enabling easy scalability.

The payload lifecycle (Figure 3) touches on every component of the VeriScale architecture, since it is the prevalent abstraction and unit of deployment in the VeriScale-enabled cloud.

1. For a detailed discussion of immutable service containers, see the link to the Immutable Service Containers project Web site in “References” on page 26.

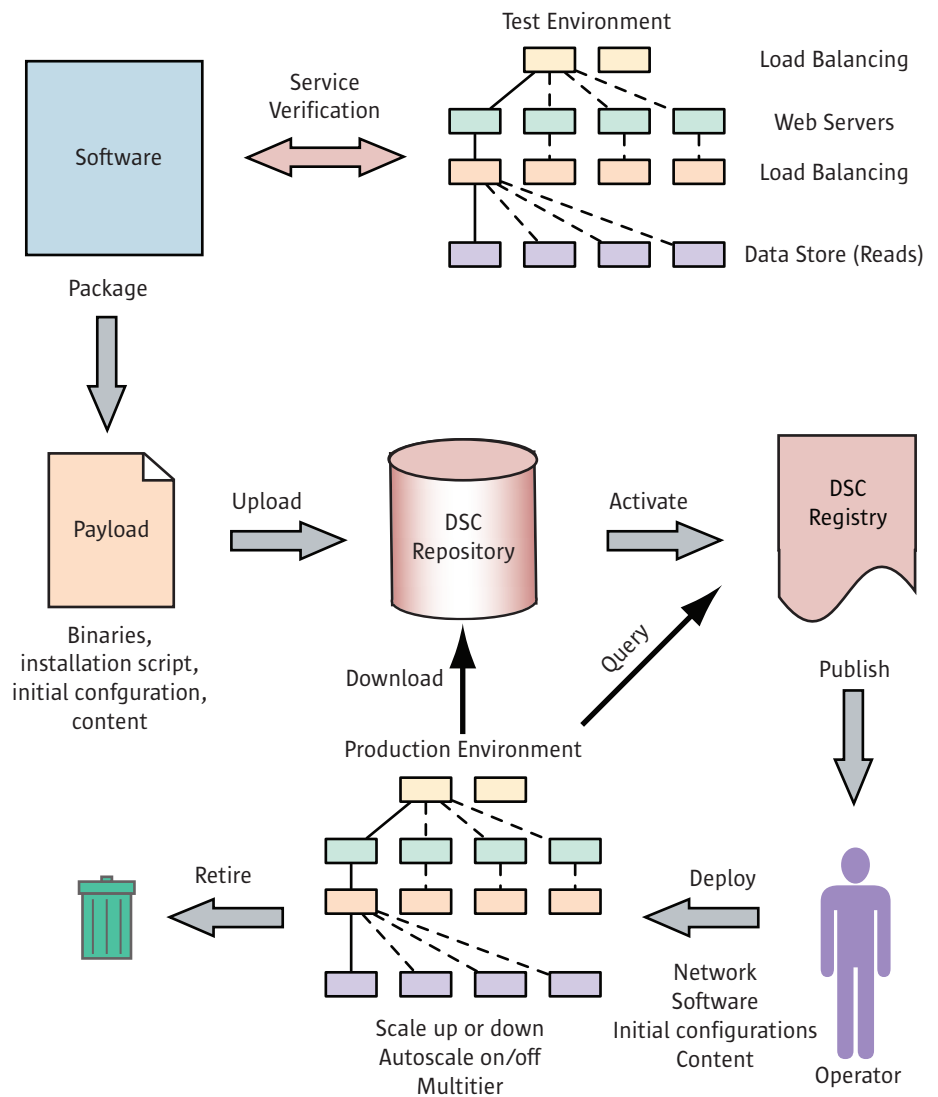


Figure 3: OpenSolaris DSC payload lifecycle

The lifecycle of a DSC payload follows seven distinct stages, described below:

1. Service Verification

In the service verification stage, the developer produces a scaled-down prototype of the desired service that is deployed to a test system. In this stage, the functional requirements and systemic qualities are verified and the service is modified and tested repeatedly until it meets its target requirements and systemic qualities.

2. Package

In the package stage following verification, the service's binaries and installation software, configuration files, and any additional required content are packaged into a single, self contained payload. To be self contained, the payload must include enough information to resolve all dependencies and not have to rely on any external source to create a functionally complete service.

3. Upload

In the upload stage, once the payload packaging is complete, the packaged payload is uploaded to the DSC repository.

4. Activate

In the activation stage, the payload's location is added and activated in the DSC registry. The DSC registry now holds all of the information needed to deploy the payload and the service encapsulated in the payload can be deployed and used.

5. Publish

Once the service is ready for deployment, the availability of the service is published by notifying the system operator that the service is available. The operator can now trigger the deployment of the service by updating the DSC registry with the number of deployed instances of the service that are required.

6. Deploy

As described in the section on "Node Controllers" on page 7, the DSC node controllers continuously query the registry for updates. Thus, when the operator updates the DSC registry, this triggers the node controllers on PODs with available resources to pull the payloads, install the software, configure it, and start the service. The DSC node controller notifies the registry that the service has been activated.

7. Retire

Once the payload has fulfilled its purpose, and there are no service instances that require it, the payload is retired. This is achieved by destroying the containers that hold the payload and removing it from the repository. This would normally be performed during an upgrade or removal of a service.

An OpenSolaris DSC Use-Case

OpenSolaris DSC helps implement a fully-automated collection of services that continuously function, normally requiring no human intervention unless a change in the deployed services is needed. This hands-off capability is illustrated in the following use-case:

- An administrator develops a payload and stores it in the repository for download. The payload is self contained and includes the required scripts to install and start the service.
- The administrator registers and activates the payload in the registry, together with its associated operational definitions. These operational definitions include how many instances of the payload should be deployed, and what its execution configuration characteristics are e.g., CPU architecture, required memory footprint, etc.
- The OpenSolaris DSC central management service creates an appropriate number of Solaris™ Zones² on the target nodes, which are then populated with payloads.
- After the initial setup is complete, a refresh-service periodically checks each payload and replaces it with another workload if needed. In this case, the refresh service updates the registry, destroys the zone that hosted the old payload, selects a new payload, and deploys it into a newly created zone.
- The registry maintains the state of the different payloads and controls the creation or destruction of payloads to maintain service levels and free resources.

2. For a detailed discussion of Solaris Zones, see System Administration Guide: Solaris Containers Resource Management and Solaris Zones linked from “References” on page 26.

Services

The SDN architecture defines a hierarchical grouping of services:

- *Service instances* are the atomic unit of functionality of a given service.
- *Service domains* are collections of service instances.
- *Service modules* are a collection of service domains of different types, each with a specific role. Together, service modules form an application.

Each service domain includes the specific attributes needed to reach it over the network. For example, a single Web server is a service instance in the service domain designed to provide a Web-serving service through a specific IP address that is common to all of the Web servers that are part of the service domain. The *Web-serving service domain* provides services as a single entity, and is comprised of a subnet in a virtual LAN that includes multiple Web servers grouped for the purpose of load-balancing. If, for example, the Web-serving service domain uses a data store service domain, together they form a service module.

Each service domain is addressable by clients that connect to the service domain and not to a specific service instance. The service domains are distinguished by their different characteristic: protocols, ports, health monitoring implementation, and security requirements.

Management

A separate management network is included in every instance of an SDN architecture. The management network enforces the security requirements of the service domains. A management domain can serve many service domains, however, a service domain can only have one management domain. The modular nature of the SDN architecture supports the addition of security modules anywhere in the architecture.

In the VeriScale architecture, the SDN provides the connection between the applications and network, and is essential to achieving network automation. The service modules in the VeriScale architecture are designed for easy replication and distribution, allowing the application to scale on demand.

Security

The SDN architecture implements security through the concepts defined by Sun Systemic Security, which is a comprehensive architectural approach to information protection and assurance. The primary principles of Sun Systemic Security are

self-preservation, defense in depth, least privilege, compartmentalization, and proportionality. For a detailed discussion of Sun Systemic Security, see the link to the Sun Systemic Security Web site in “References” on page 26.

Security in the SDN architecture is implemented through security modules and security domains that secure specific service modules or service domains respectively. Security modules or security domains can implement any combination of security functionality for the underlying services they secures. For example, a firewall security domain can protect a specific Web-serving service domain. A firewall security module, consisting of any number of firewall security domains, can protect a Web application service module.

Security classification labels are used to distinguish the various service domains from each other depending on their security requirements. The security classification labels logically represent the different service domains with common security requirements. For example, the security classification labels can be used to map a service domain to its management domain.

Chapter 6

Network Optimization Through Distribution

Distributed processing is a core concept in the VeriScale architecture. By avoiding the reliance on centralized resources and the distribution of processing across a large number of network nodes, the processing capacity of a datacenter is better utilized, while reducing the required network capacity. In the following sections, the practical ways that VeriScale optimizes deployment and load-balancing are investigated in further detail.

Deployment Optimization

When a Web server is deployed in a traditional datacenter, a provisioning server connects to the target node, uploads the Web server application, performs the installation, and uploads content. Manual intervention or difficult-to-maintain, semi-automatic scripts are needed to configure the network appropriately for the added service. If there is a sudden need to allocate several dozen, or for that matter, several thousand Web servers, the traditional datacenter would struggle to service the request because it depends on centralized resources.

In the VeriScale architecture, the act of allocating the POD to the Web-serving service domain causes the POD to pull the appropriate Web server payload from the repository. For this purpose, the POD uses the DSC registry to determine all of the necessary network information it needs. With the exception of the DSC registry, the POD can provision itself without using centralized, potentially scarce resources, nor does it need manual intervention. Assuming the PODs are available and activated, servicing the request for thousands of new Web servers is virtually as easy as servicing the request for a single Web server.

Load-Balancing Optimization

The flexibility needed for a dynamic, scalable, load-balanced environment can be achieved by using indirection and resource pooling to abstract the destination address. This can be demonstrated by describing a Web application deployed on a traditional network architecture, utilizing OpenSolaris DSC and the VeriScale architecture.

Load-Balancing on a Traditional Architecture

Traditional load-balanced networks (Figure 5), are normally implemented with a central load balancer that redirects traffic. Clients only need to know about the load balancer, and not the final destination of their requests. This approach results in sub-

optimal routing since the data is routed through the load balancer even if there is a shorter route to the destination. In addition, the load balancer must be replicated for the system to scale.

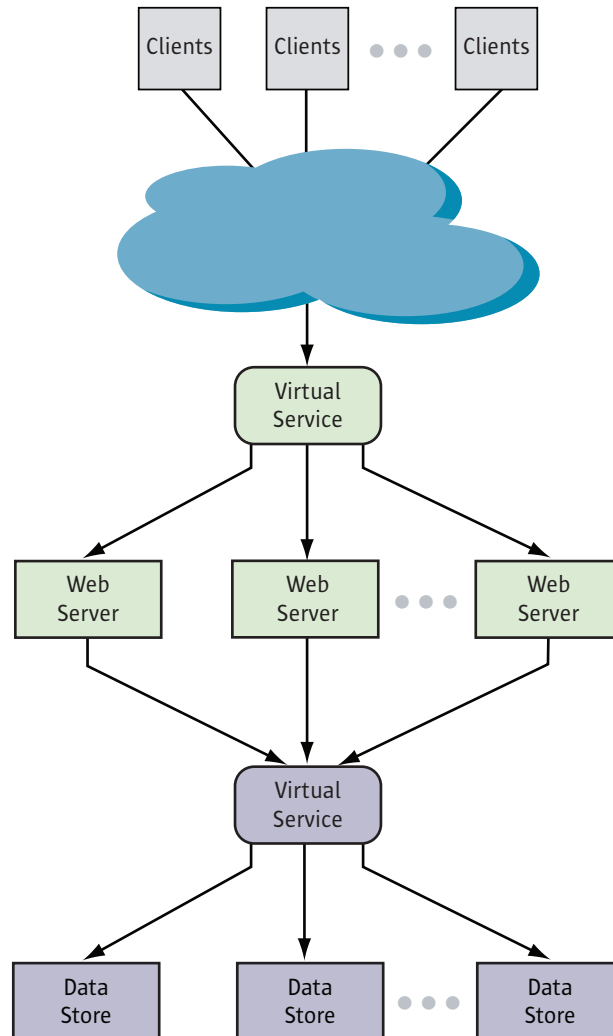


Figure 5: A traditional load-balanced Web server application

Load-Balancing Utilizing OpenSolaris DSC

With OpenSolaris DSC-based load-balancing (Figure 6), when a client connects to a Web-serving virtual service — an instance of a “service domain” in SDN terminology — through a virtual address, the load balancer redirects the request to a server — an instance of a “service instance” in SDN terminology — for the Web-serving service domain. The service domain functions as a resource pool designed to service requests of this type. Scaling is implemented by adding service instances to the

service domain while the node controller automatically queries the registry for the information it needs. This approach lets the load balancers automatically update their list of service instances from the list maintained in the DSC registry.

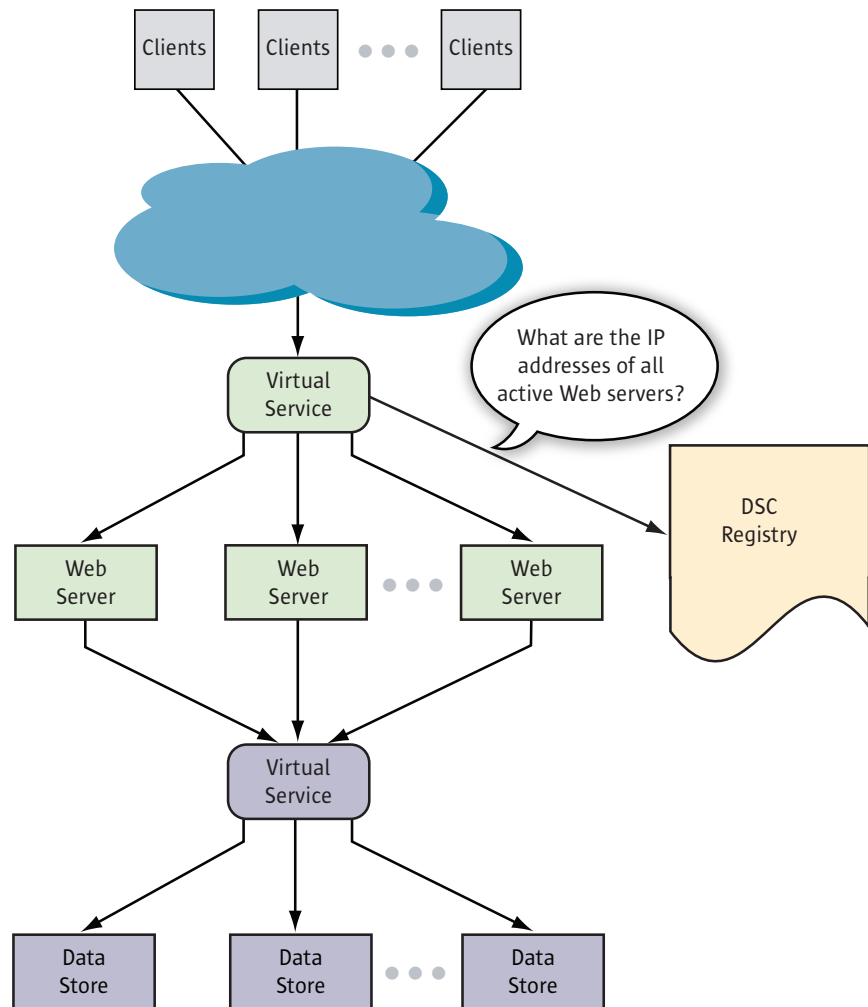


Figure 6: OpenSolaris DSC-based load-balanced Web server application

Load-Balancing on a VeriScale Architecture

In the VeriScale architecture, the load-balancing scenario (Figure 7) is further optimized by implementing the networking logic locally in the service instance's containers and treating the networking logic as part of the application. This implementation makes the POD self-contained in terms of network load-balancing logic. As payloads are added to the POD, the load-balancing logic embedded within each payload allows the load-balancing functionality of the POD to scale up or down as needed. In turn, the load-balancing functionality of the service domain to

which the POD is assigned is also able to scale up or down. Extending this principle by colocating other heavily-used network services with their client applications can further improve network performance and scalability.

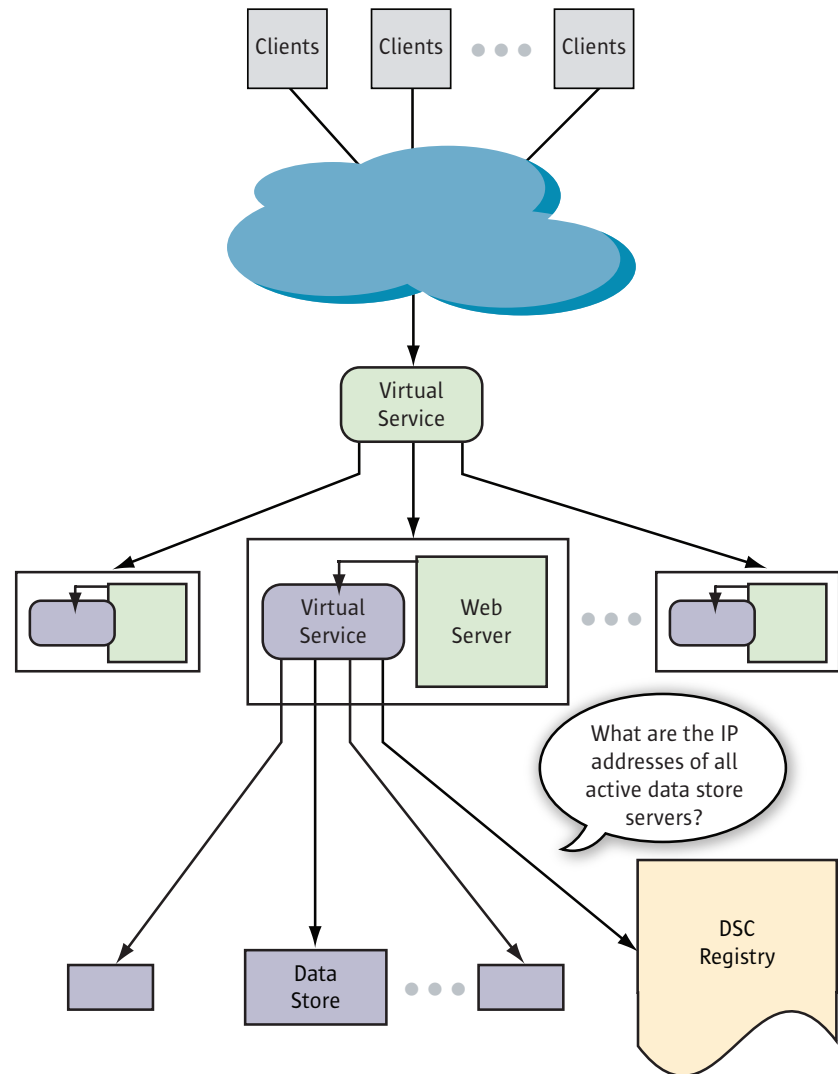


Figure 7: VeriScale load-balanced Web server application

A Web application — an instance of a “service module” in SDN terminology — consisting of a Web-serving service domain and a data store service domain can illustrate the benefit of having a self-contained container. When the Web server requires a service from the data store, the Web server communicates the request to the data store load-balancing component located in the same container as the Web server. The embedded load balancer redirects the request to the appropriate data store service instance either in the same POD or in another POD. In addition

to optimizing network performance, embedding the load balancer with the Web server in this way allows the load-balancing networking logic in the POD to scale automatically as containers are added when scaling the Web application.

Network Data Path Optimization on a VeriScale Architecture

Colocating the load balancers with the service instances that use them creates an additional significant opportunity for network optimization. The load balancer measures the response times and optimizes the choice of destination service domains based on their responsiveness on both the networking and application levels. When optimizing network data paths in this way (as illustrated in Figure 2 on page 6) the network end-points evaluate the optimal path to their network peer according to the peer's location:

- When both end-points are on the same host, they use the Network Virtualization and Resource Control — OpenSolaris Project Crossbow network virtualization stack (blue path)
- When both end-points are on different hosts in the same rack, they use the top-of-rack switch (green path)
- When both end-points are on different racks, they use the network backbone switch (red path)

As the services on the network scale, network bandwidth becomes an increasingly scarce resource. Incorporating network data path optimization into the load-balancing logic saves on bandwidth and improves scalability.

Benefits of VeriScale Optimization Through Distribution

Colocating software components that implement networking logic with related applications reduces the need for over-provisioned centralized resources, helps reduce network traffic, and reduces the number of physical devices on the network. At the same time, latency can be improved as there are fewer devices, fewer communication hops, less traffic, and optimized communication paths.

Chapter 7

Proof of Concept for the VeriScale Architecture

A POC was implemented to show the feasibility of the VeriScale architecture. The POC consisted of a service module deployed with two service domains, each with a large number of service instances. This configuration was deployed onto a set of target nodes, while using a minimal amount of centralized resources, to achieve rapid scale deployment of the architectural model.

The functionality of the architectural model was tested in a virtual environment prior to packaging the payloads per the stages described in the section “Payload Lifecycle” on page 8.

Objective

The objective of this POC was to use the VeriScale architectures to deploy and dynamically scale a two-tier service module with a Web-serving service domain and a data store service domain. Both service domains were required to scale individually on demand. Each of the service modules was designed to use software load balancers to automate generation, expansion, and contraction of the service instance pools and to automatically deploy, install, and execute software payloads.

The POC was designed to demonstrate that the DSC network automation software can:

- Automatically scale pools of up to 600 Web server service instances and up to 400 data store service instances on at least 1000 Solaris Zones on 15 servers
- Automatically set up the network, deploy the services, and activate them
- Deploy on a heterogeneous environment with both Solaris 10 update 7 and OpenSolaris running on a mixed pool of SPARC® and x86 servers

Platform

Software, hardware, and networking components used in the POC are described in the sections that follow.

Software

The software platforms used in the POC are described in Table 1.

Table 1. Software platforms used in the POC.

Function	Platform	Vendor
Web server	Apache HTTP Servers 2.2	Apache Software Foundation
Data store	MySQL™ Enterprise Server 5.1	MySQL AB, Sun Microsystems, Inc.
OS	OpenSolaris build 117	Sun Microsystems, Inc.
OS	Solaris 10 update 7	Sun Microsystems, Inc.
Load balancer	ZXTM LB 5.1	Zeus Technology Ltd.
HTTP load generator	ZeusBench 5.1	Zeus Technology Ltd.

Servers

Servers used in the POC are described in Table 2.

Table 2. A range of SPARC and x64 servers were used in the POC.

Function	Quantity	Server	Architecture	# threads	Clock (GHz)	Memory (GB)	Disks
DSC registry and repository	1	Sun Fire™ X4600 Server	x64	32	2.7	64	4
DSC nodes	4	Sun SPARC Enterprise® T5220 Server	SPARC	64	1.4	64	4
DSC nodes	6	Sun SPARC Enterprise T5440 Server	SPARC	128	1.4	64	4
DSC nodes	2	Sun Fire X4450 Server	x64	24	2.7	24	4
DSC nodes	3	Sun Fire X4600 Server	x64	32	2.7	64	4

Solaris OS was installed in the non-global zones on the target nodes.

Network

The systems were deployed on two networks: a management network and an application network.

The network switches were three Force10 S50 48 port switches from Force10 Networks, Inc. The network switches were pre-configured with a set of VLANs available for use on each port and with VLAN tagging enabled.

Setup

The POC system consisted of the following components:

- A single server hosting the DSC central components — registry, repository, and node controller
- Fifteen servers as target nodes to ultimately host the service instances, each with a node controller
- Software load balancers that were manually installed on four of the servers to serve as front-end load balancers in a clustered deployment
- A node controller for the front-end load balancers that poll the registry for updates of the number of Web servers on the network for the virtual service configuration
- An HTTP load generator to generate client HTTP requests
- Network switches pre-configured with a set of VLANs available for use on each port
- Two VLANs — one for the Web-serving service domain, the other for the data store service domain — each on its own subnet
- The data store payload, including a single table of names
- The Web server payload, including an HTTP server, a software load balancer, and the node controller
- The HTTP server included configuration files and minimal content
- The load balancer included the initial configuration of the virtual services
- The node controller for the load balancer was configured to query the registry for updates of the number of data stores available on the network

Execution

The POC execution was triggered by entering 600 and 400 in the registry, as the number of required Web server instances and required data store instances respectively. From this point on, the requested instances deployed automatically.

The Web server instances, once deployed, were configured to download additional content from the service colocated on their target node. This service was provided by the software load balancer that was installed to listen on the data store request port through the network loopback interface.

Within six minutes at least one of the Web server instances and one of the data store instances were deployed and both service domains were available to service requests. The data from the data store service domain was available to the Web-serving service domain, while the Web-serving service domain was available to

service external client requests. Within 60 minutes, all 600 Web server service instances and all 400 data store service instances were available to service requests, with access to the full content.

A node controller queried the registry every three minutes for updates and changes, including the addresses of the Web server data store instances that became available to service requests. When new servers were identified, they were added to the pool of resources for the service domains by updating the load balancer configuration.

The VeriScale Concept Proved

The POC proved that the VeriScale architecture is able to deliver its core requirements of automatic and efficient deployment of applications, scaling to many hundreds or potentially thousands of instances across a datacenter.

Chapter 8

The VeriScale-Enabled Business Process

The VeriScale architecture supports the software application definition, development, and deployment business process as illustrated in Figure 8.

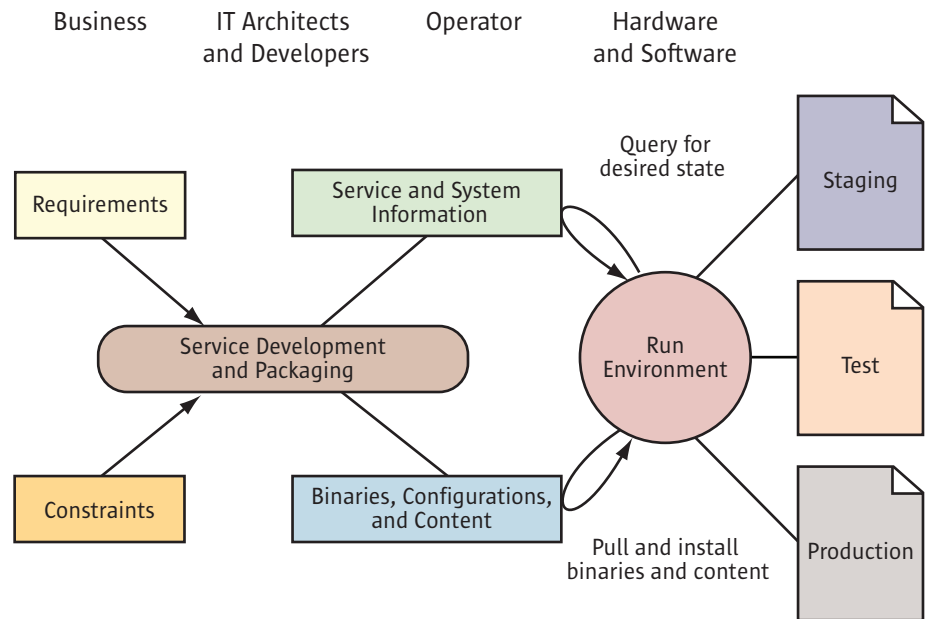


Figure 8: The VeriScale business process

The business process consists of the following phases:

- The organization's IT business stake-holders define the business requirements and constraints for the applications.
- The IT architects and developers use the requirements and constraints to develop the applications. During the development process, the developers use the VeriScale architecture for staging and testing by creating service definitions registered with the service registry and the packaged application stored in the payload repository as development payloads.
- Once the development is complete and verified, the applications are stored in the production payload repository and the services are registered with the production service registry.

Using an identical environment to develop services and then deploying to production helps ensure that the services are ready for deployment in the production environment upon completion of the development cycle. In addition, the same environment can be used for both development and production, contributing to the cost-efficiency of the IT infrastructure.

Chapter 9

Conclusion

The primary requirements of the VeriScale architecture are scalability, flexibility, and efficiency, and the primary mechanism the VeriScale architecture uses to deliver on these requirements is POD and container self-sufficiency. Self-sufficiency is achieved, to the extent possible, by encapsulating each service within a payload with the full range of capabilities it requires, while minimizing the need for external, centralized resources. Once delivered to any suitably-capable container in a POD, these payloads — applications, application platforms, or entire VMs — can configure themselves and provide the useful function they were designed to provide with minimal support from central resources. In addition, self-sufficiency provides additional benefits including network optimization and improved datacenter cost-efficiency.

These capabilities help create cost effective, elastic service domains that can rapidly scale up or down on demand, limited only by the availability of hardware resources.

About the Author

Mikael Lofstrand is currently the Chief Technologist of Networking Technologies for Sun's Global Sales and Services organization and is part of the Chief Architects Office. Mikael is an expert in design, implementation, and management of datacenter networking technologies, he has more than 16 years of experience in the IT industry. He is renowned for his work on Sun's Service Delivery Network architecture as well as his contribution in Sun's Dynamic Infrastructures and network strategies for networking centric datacenters.

Acknowledgments

The author would like to recognize the following contributors without whom this paper would not be possible:

- Jason Carolan for working together with the author developing the VeriScale architecture, providing feedback on this paper, and for the countless calls over the years that resulted in this architecture.
- Robert Holt for his outstanding work on the code for OpenSolaris DSC, providing feedback on this paper, and for his hard work driving the POC.
- Glenn Brunette for his insights, his inspiration, his invaluable feedback, and for always being available for discussions.
- Miles Martin at Zeus Technologies for insights on network load-balancing software.

References

Table 3. References for more information.

Web Sites	
VeriScale Architecture	http://wikis.sun.com/display/VeriScale
Network Virtualization and Resource Control — OpenSolaris Project Crossbow	http://opensolaris.org/os/project/crossbow/
Project OpenSolaris Dynamic Service Containers	http://wikis.sun.com/display/DSC
Sun's Pattern-based Design Framework: The Service Delivery Network	http://wikis.sun.com/x/_l76
The Service Delivery Network — A Case Study	http://wikis.sun.com/x/so76
Datacenter Reference Guide	http://wikis.sun.com/x/XA89Ag
Free Security Hardened Virtual Machine Image	http://blogs.sun.com/gbrunett/entry/free_security_hardened_virtual_machine
Immutable Service Containers	http://kenai.com/projects/isc
System Administration Guide: Solaris Containers Resource Management and Solaris Zones	http://docs.sun.com/app/docs/doc/817-1592/
Sun Systemic Security	http://sun.com/security
Sun's Modular Datacenter	http://sun.com/products/sunmd/s20/
Sun BluePrints Articles	
Sun's Pattern-Based Design Framework: The Service Delivery Network	http://sun.com/blueprints/0905/819-4148.pdf

Ordering Sun Documents

The SunDocsSM program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The docs.sun.com Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is <http://docs.sun.com>

To reference Sun BluePrints Online articles, visit the Sun BluePrints Online Web site at: <http://www.sun.com/blueprints/online.html>

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN (9786) Web sun.com

© 2009 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, MySQL, OpenSolaris, Solaris, Sun BluePrints, Sun Fire, SunDocs, and Sun SPARC Enterprise are trademarks or registered trademarks of Sun Microsystems, Inc., or its subsidiaries in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Information subject to change without notice. Printed in USA 09/09

