Hands-On Linux for Architects

Designing and implementing Linux based IT solutions

Packt>
www.packt.com

By Denis Salamanca and Esteban Flores

What this book covers

`Chapter 1`, *Introduction to design methodology*, covers different stages of architecting a solution and why each part is important.

`Chapter 2`, *Defining GlusterFS Storage*, helps you understand the decisions needed to be made when deploying a high performance storage solution using gluster.

`Chapter 3`, *Architecting a storage cluster*, helps you explore the design aspects of implementing a clustered solution using GlusterFS and its various components.

`Chapter 4`, *Using GlusterFS on cloud infrastructure*, will explain the configuration necessary to implement a GlusterFS on different environments like cloud or on prem.It will also explain how to configure the underlying storage and its tunables required for best performance.

`Chapter 5`, Analyzing Performance in a Gluster system, will go through explaining the accomplished configurations as well as testing to verify the implementation to be working and delivering the performance as expected.

`Chapter 6`, *Creating a Highly available self-healing Architecture*, covers a scenario where our fictional customer will present us with the problem that he has and his expectations for a possible solution, he currently leverages Microsoft's public cloud Azure, and he wants to take advantage of it provisioning his solution on the cloud.

`Chapter 7`, *Understanding the core components of a Kubernetes cluster*, will go over the core kubernetes components giving a 1,000ft view of each one and how they will help us solve our customers problem.

`Chapter 8`, *Architecting Kubernetes on Azure*, will help us start the design stage of our kubernetes based solution in Azure Kubernetes Services, considering the use of each of its components and how will they help us achieve our goal.

`Chapter 9`, *Deploying and configuring Kubernetes for your NGINX application*, will help us implement our design step by step, by creating the azure resources using Azure CLI for Linux all the way to creating each Kubernetes component that will help us achieve our final solution that will provide High availability, Self Healing and Load balancing for a NGINX web application

`Chapter 10`, *Monitoring with ELK stack*, will explain what each component of ELK does. ELK stack will help simplify this task by allowing the logs and metrics from different sources to be aggregated in a single indexable location, Elasticsearch.

`Chapter 11`, *Designing an ELK Stack*,will go through the design considerations when deploying an ELK stack.

`Chapter 12`, *Using Elasticsearch, Logstash, and Kibana to Manage Logs*, helps us to look at some considerations to be taken when deploying to achieve the best results.

`Chapter 13`, *Centralized Linux Patch Management with Spacewalk*, will help us check the needs for you to having a centralized management utility for your linux infrastructure.

`Chapter 14`, *Lets take a Spacewalk*, will provide an overview of all the different components of the Spacewalk solution and the different roles they play.

`Chapter 15`, *Designing a simple but powerful Spacewalk solution*, we will propose a design for a Spacewalk solution, and we will understand why it's simple but also offers an incredible amount of functionality.

`Chapter 16`, *Implementing a Spacewalk Infrastructure*, we will go through the process that requires to stand up a Spacewalk infrastructure in order to manage our systems in a centralized fashion to achieve maximum efficiency.

`Chapter 17`, *Design Best Practices*, will help us go through the design best practices, the dos and don'ts. We'll focus on understanding the what works best for the solutions we explored before.

Table of Contents

Preface

When it comes to an architectural job role, that particular individual should be able to understand any infrastructure and should be capable of designing an efficient environment. This book will help you understand and achieve the level of knowledge required to architect and implement various IT solutions based on Linux.

Introduction to Design Methodology

These days, IT solutions require increased performance and data availability, and designing a robust implementation that meets these requirements is a challenge that many IT experts have to go through everyday.

In this chapter, you will learn the basics from a 10,000 ft view of architecting IT solutions in any type of environment, from virtualized infrastructure, bare metal, and to even public Cloud, as basic concepts of solution design apply for any environment.

You will explore the following subjects:

- Defining the stages of solution design and why they matter
- Analyzing the problem and asking the right questions
- Considering possible solutions
- Implementing the solution

Correctly understanding the aspects that you need to consider when architecting a solution is very crucial for the success of the project, as this will determine which software and hardware, and which configuration will help you achieve the desired state that meets the needs of your customers.

Defining the stages of solution design and why they matter

Like many things, designing solutions is a step-by-step process that not only involves the technical aspects, nor necessarily the technical parties. Usually, you will be engaged by an Account Manager, Project Manager or, if you are lucky, a CTO, who understands the technical part of the requirements. They are looking for an expert who can help them deliver a solution to a customer. These requests usually do not contain all the information you will need to deliver your solution, but it's a start to understand what your goal is.

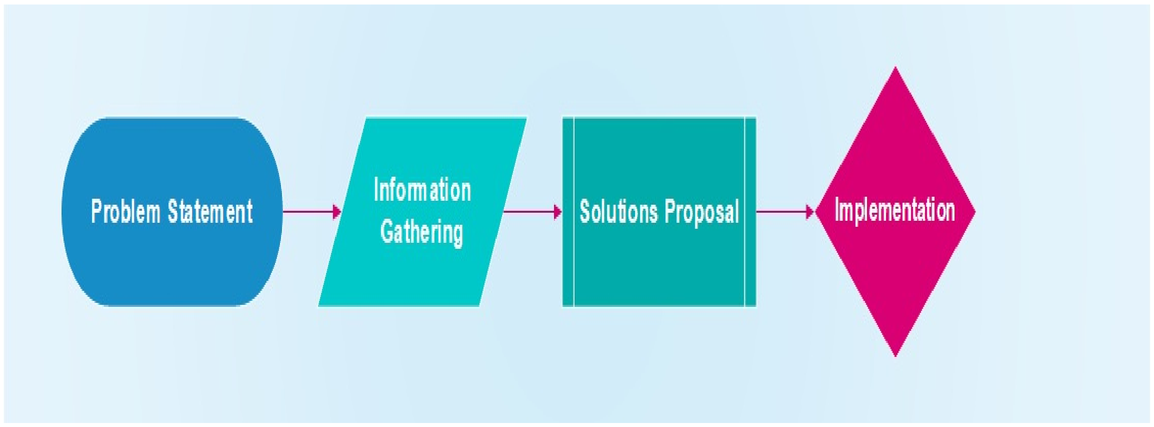
For example, you receive an email from a Project Manager with the following statement:

"We require a solution that can sustain at least 10 thousand website hits and stays available during updates as well as survives outages. Our budget is considerably low, so we need to spend as little as possible, with little to no upfront cost. We're also expecting this to gain momentum during the project's lifecycle."

From the previous statement, you can only get a general idea of what is required, but no specifics have been stated. So, you can only make assumptions like "We require a solution that can sustain at least 10 thousand website hits", which, for a design, are not good enough, as you require as much information as possible to be able to resolve the problems exposed by your customer. This is where you have to inquire for as many details as possible, to be able to provide an accurate set of proposals for your customer, which will be the first impression your customer will have of the project. This part is critical, as it will help you understand if you have your customers' vision right.

It is also important to understand that you need to deliver several different solutions for the customer, as the customer is the one who decides which one fits their business needs the most. Remember that each solution has its own advantages and disadvantages. After the customer decides which way to go, you will have what is necessary to move to the implementation of your proposal, which can always trigger more challenges. It will require, more often than not, some final personalized tuning or changes that were not considered in the initial proof of concept.

From our previous analysis, you can see four well-defined stages of the process that you need to follow in order to reach the final delivery:



There are many more stages and design methodologies that we could cover, but since they're not in the scope of this book, we will be focusing on these four general ones to help you understand the process in which you will be architecting your solutions.

Analyzing the problem and asking the right questions

After getting the initial premise, you need to dissect it into smaller pieces in order to understand what is required. Each piece will raise different questions that you will ask your customers later. These questions will help fill in the gaps for your proofs of concept, always ensuring that your questions cover all business needs from all view standpoints: the business standpoint, the functional standpoint and, finally, the technical standpoint. One good way to keep track of what questions arise and which business need they will be resolving is to have a checklist that contains from which standpoint the question is being asked and what is resolving or answering.

It is also important to note that as questions become answers, they can also come with constraints or other obstacles, which will also need to be addressed and mentioned during the Proof of Concept stage. The customer will have to agree with them and will be decisive for them when selecting the final solution.

From our previous example, you can analyze the premise in the following way by dissecting it into standpoints:

"We require a solution that can sustain at least 10 thousand website hits and stays available during updates as well as survive outages. The budget is considerably low, so we need to spend as little as possible, with little to no upfront cost. We're also expecting this to gain momentum during the project's lifecycle."

Technical standpoint

From this perspective, we will analyze all technical aspects of the premise, anything that you will need to provide the initial technical requirements of your solution.

We will analyze it in the following way:

- You can understand, from the premise, that your customer needs some kind of solution that can sustain some amount of website hits, but you can't be certain if the web server is already set up, and what the customer needs is only a load balancing solution. Alternatively, maybe the customer needs both, a web server, that is, NGINX, Apache, or something of that sort, and the load balancing solution.
- The customer mentions at least 10 thousand hits to their website, but he didn't mention if these hits are concurrent per second, daily, weekly, or even monthly.
- You can also see that they need to stay available during updates and be able to continue serving their website if the company has an outage, but all these statements are very general, since availability is measured in 9s. The more 9s you have, the better! (in reality, this is a percentage measurement of the amount of time during the year; a 99.% availability means that there can only be

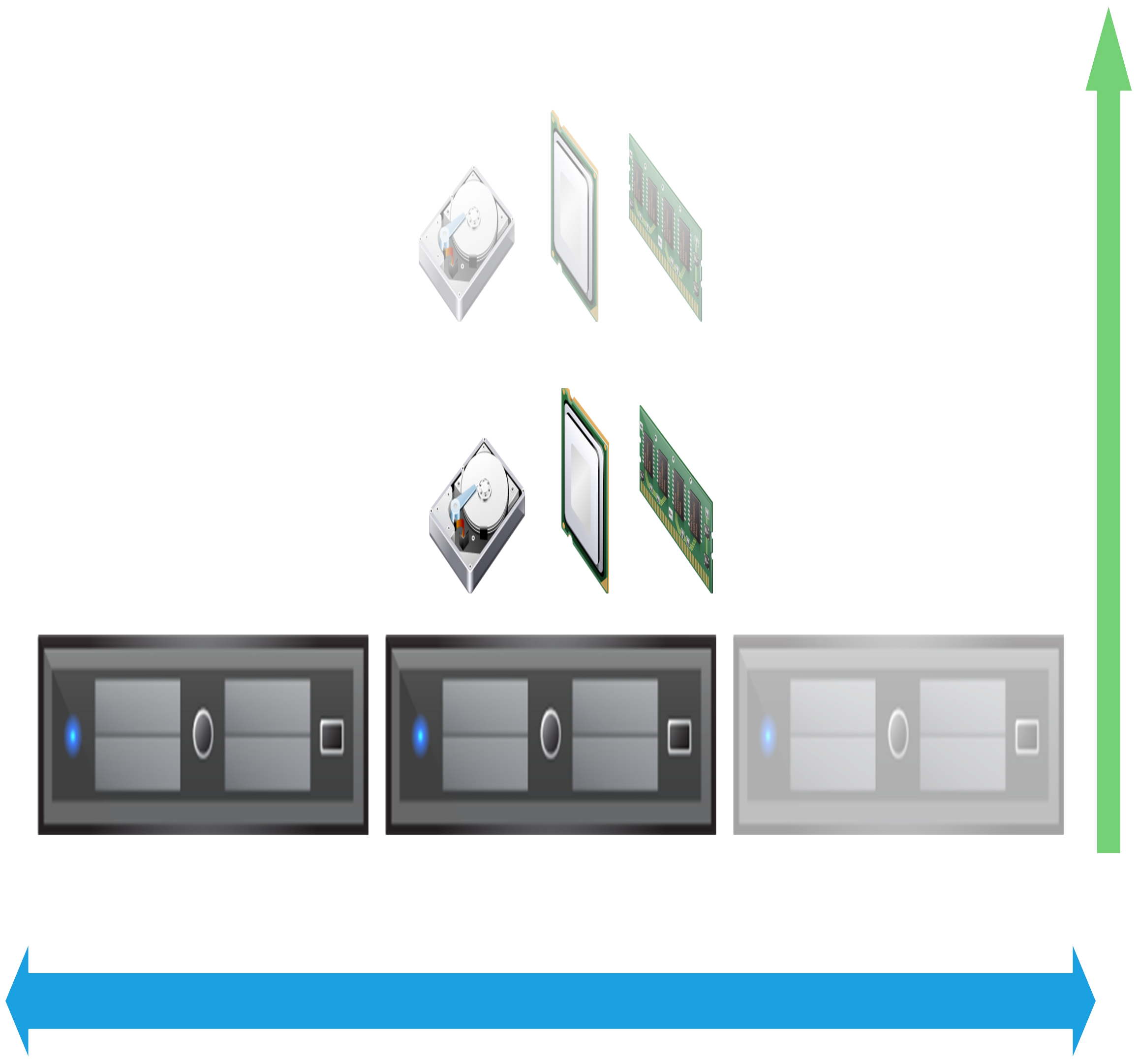
526 minutes of downtime per year). Outages are also very hard to predict, and it's almost impossible to say "I will never have an outage"; therefore, you need to plan for it. You have to have an RPO (Recovery Point Objective) and RTO (Recovery Time Objective) for your solution in case of a disaster. The customer didn't mention this, and it is crucial to understand how much time a business can sustain an outage.

- When it comes to budget, this is usually from a Business Perspective, but the technical aspects are affected directly by it. Looks like the budget in the project is tight, and the customer wants to spend as little as possible in their solution, but they're not mentioning exact numbers, which you require in order to fit your proposals to it. "Little to no upfront cost?" What does this mean? Are we repurposing the existing resources and building a new solution? How can we implement a design with no upfront cost?
  One way to overcome low budgets or no upfront cost, at least in software, is to utilize OSS (Open Source Software), but this is something that we need to ask the customer.

- Gaining momentum can only mean that they are predicting that their user base will grow eventually, but you need numbers of how much they predict this will grow, and how fast. This will imply that you have to leave the solution ready to be scaled vertically or horizontally.

> Vertically, by leaving space to increase the resources eventually and taking into account the businesses' procurement process if you need to buy more resources like RAM, CPU, Storage, or such. Horizontally will also involve a procurement process and a considerable amount of time to integrate a new node/server/VM/Container into the solution. None of these is included in the premise and its vital information.

Here, we have a comparison of horizontal and vertical scaling. Horizontal scaling adds more nodes, while vertical scaling adds more resources to the existing nodes:



Examples of questions that you could ask to clarify the gray areas:

- Is this solution for a new/existing website or web server?
- When you say '10 thousand hits', are these concurrent per second or daily/weekly/monthly?
- Do you have any estimates or current data of how large your user base is?
- Considering that the budget is low, can we use Open

Source Software?

- Do you have the technical resources to support the solution in case we use OSS?
- Do you have any sort of update infrastructure in place or version control software implemented already?
- When you say "Little to no upfront cost", does this mean that you already have hardware, resources, or infrastructure (Virtual or Cloud) available that we could recycle and/or reuse for our new solution?
- Are there any Disaster Recovery Sites in place that we could use to provide High Availability?
- If your user base grows, will this generate more storage requirements or only compute resources?
- Do you plan on performing any backups? What is your backup scheme?

From the technical perspective, once you start designing your POCs (Proofs of Concept) more questions will arise based on the software or hardware that will be used in the solution. You will need to know how they fit or what is needed for them to adjust to the customer's existing infrastructure, if any.

Business standpoint

Here, we will be analyzing the statement from a business perspective, taking into account all the aspects that can affect our design:

- A main requirement is performance, as this affects how many hits the solution can sustain. Since this is one of the main objectives of the solution, it needs to be sized to meet business expectations.
- Budget seems to be a main constraint that will affect the project's design and scope.
- There is no mention of the actual available budget.
- Availability requirements affect how the business should react in case of an outage. As there's no specific SLA (Service Level Agreement), this needs to be clarified to adjust to the business needs.
- A main concern is the upfront cost. This can be lowered considerably by utilizing OSS (Open Source Software), as there are no licensing fees.
- It is mentioned that the solution needs to remain up during maintenance operations. This might indicate that the customer is willing to invest in maintenance operation for further upgrades or enhancements.

- The statement "We're also expecting this to gain momentum" indicates that the solution will change in the amount of resources needed, thus directly affecting the amount of money consumed by it.

Things to look for when clarifying doubts from a business standpoint:

- Based on the performance requirements, what is the business impact when performance goes below the expected baseline?
- What is the actual budget for the project?
- Does the budget take into account maintenance operations?
- Considering the possible unplanned outages and maintenance, how much time exactly can your website be down per year? Will this affect business continuity?
- If an outage happens, how much time can the application tolerate not receiving data?
- Do we have data of any sort from which we can estimate how much your user base will grow?
- Do you have any procurement process in place?
- How much time does it take to approve the acquisition of new hardware or resources?

Functional standpoint

In the functional standpoint, you will be reviewing the functional side of the solution:

- You know that the customer requires 10 thousand hits, but which types of users will be using this website?
- You can see that it requires 10 thousand hits, but the premise does not specify what the user will be doing with it.
- The premise states that they need the solution to be available during updates. By this, we assume that the application will be updated, but how?

To clarify the gaps in the functional standpoint, we can query for the following information:

- What type of users will be consuming your application?
- What will your users be doing in your website?
- How often will this application be updated or maintained?
- Who will be maintaining and supporting this solution?
- Will this website be for internal company users or

external users?

It is important to note that functional standpoint overlaps considerably with the business standpoint, as they are both trying to address similar problems.

Once we have gathered all the information, you can build up a document summarizing the requirements of your solution; ensure that you go through it with the customer and that they agree of what is required to consider this solution complete.

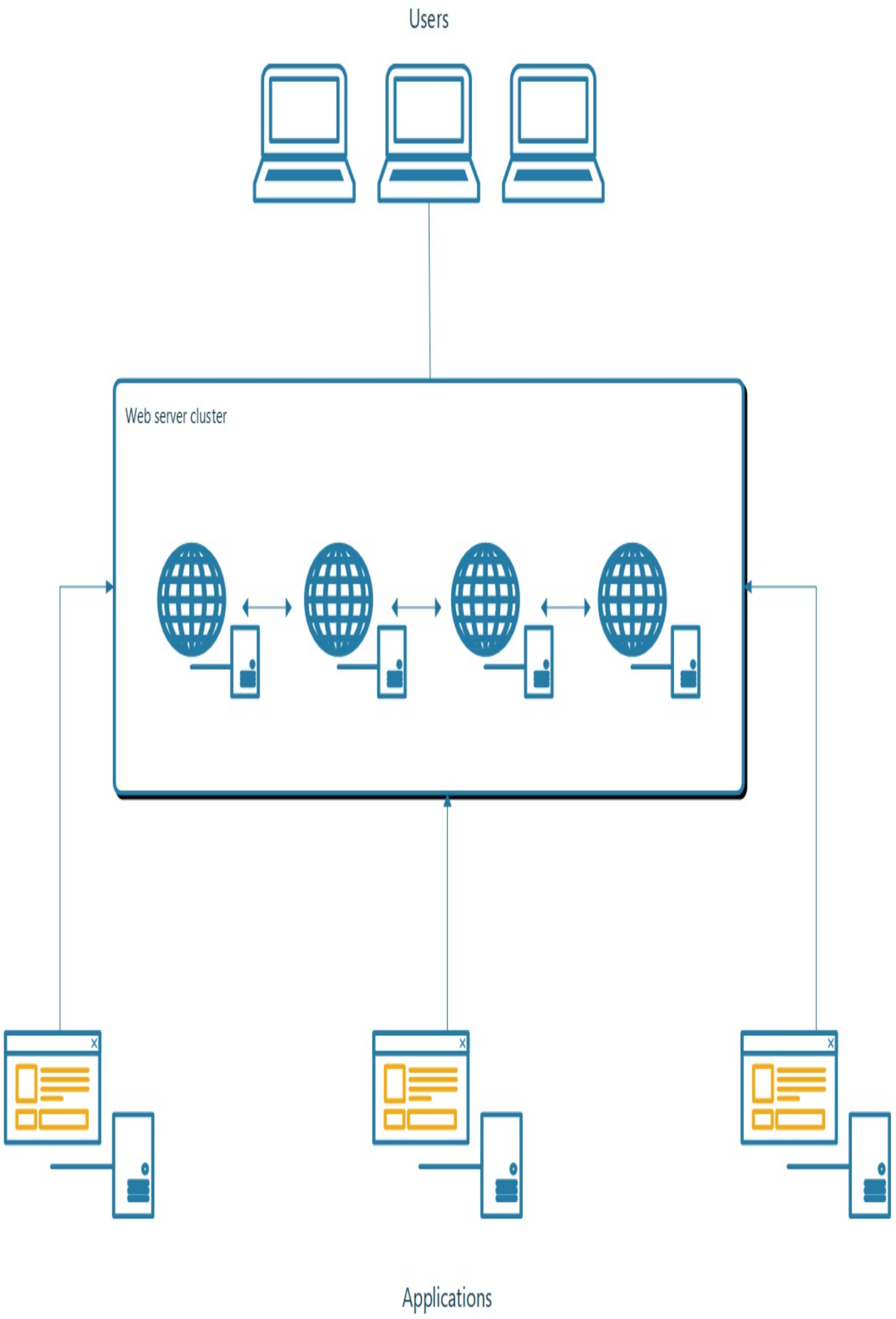
Considering possible solutions

Once all the doubts that arose during the initial premise have been cleared, you can move on and construct a more elaborate and specific statement that includes all the information gathered. We will continue working with our previous statement, and assuming that our customer responded to all of our previous questions, we can construct a more detailed statement, as follows:

"We require a new web server for our financial application that can sustain at least 10 thousand web hits per second from our approximately 2,000 users, alongside other 3 applications that will consume its data. It will be capable of withstanding maintenance and outages through the use of high availability implementations with a minimum of four nodes. The budget for the project will be $20,000 for the initial implementation, and the project will utilize Open Source Software, which will lower upfront costs. The solution will be deployed in an existing virtual environment, whose support will be handled by our internal Linux team and updates will be conducted internally by our own update management solution. The user base will grow approximately every 2 months, which is within our procurement process, allowing us to acquire new resources fairly quickly, without creating extensive periods of resource contention. User growth will impact mostly compute resources."

As you can see, it is a more complete statement on which you can already start working. You know that it will utilize an existing virtual infrastructure. Open Source Software is a go, high availability is also required, and it will be updated via an update and version control infrastructure that it is already in place; so, possibly, only monitoring agents will be needed for your new solution.

A very simplified overview with not many details of the possible design can be the following:


Users
Web server cluster
Applications

In the diagram, you can see that it's a web server cluster that provides high availability and load balancing to the clients and applications that are consuming the solution.

As you are already utilizing much of the existing infrastructure, there are a lot less options for possible proofs of concept, so this design will be very straightforward. Nonetheless, there are certain variables that we can play with to provide our customer with several different options. For instance, for the web server, we can have a solution with Apache and another with NGINX, or a combination of both, having Apache hosting the website and NGINX providing load balancing.

Proof of concept

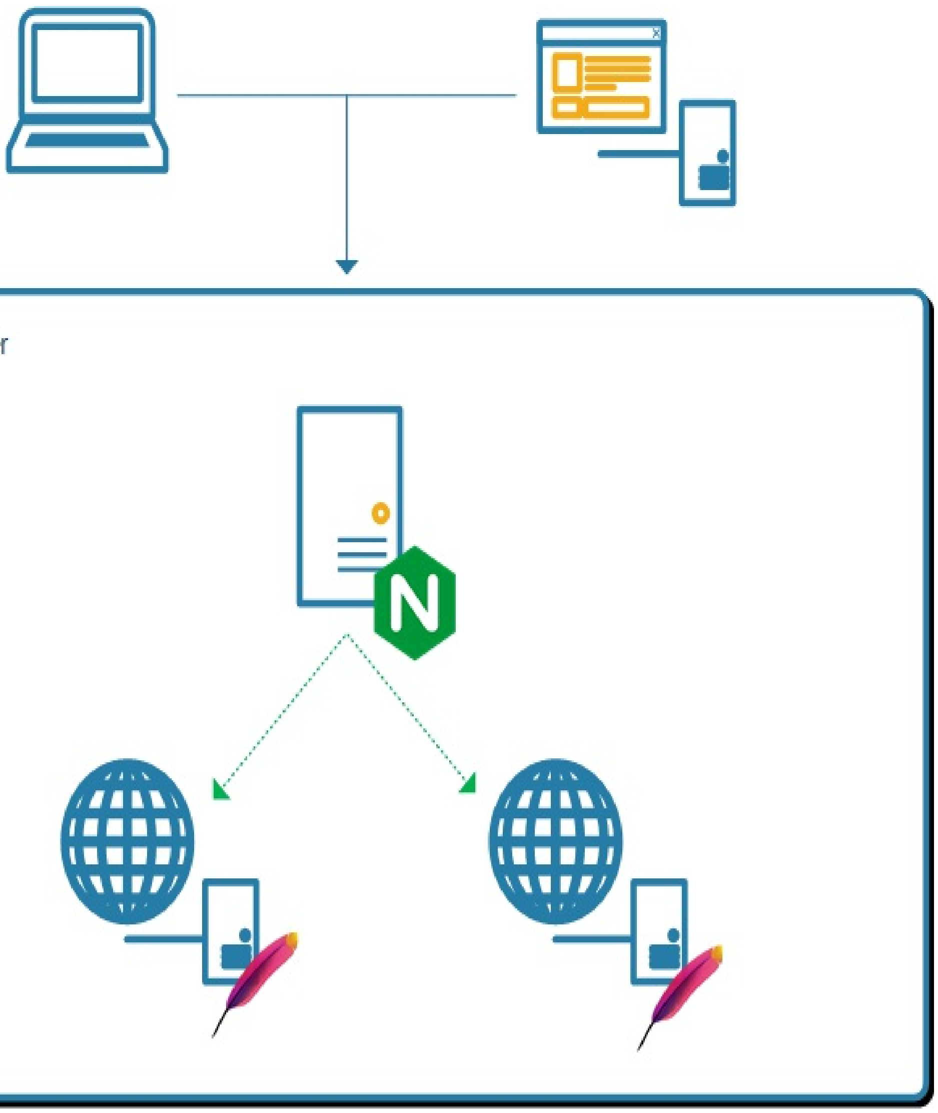
With a complete statement and several options already defined, we can proceed to provide a proof of concept based on one of the possible routes.

A proof of concept is the process of demonstrating an idea or method, in our case a solution, with the aim of verifying a given functionality. Additionally, it provides a broad overview of how the solution will behave within an environment, allowing further testing to be able to fine-tune for specific workloads and use cases.

Any proof of concept will have its advantages and disadvantages, but the main focus is for customers and architects to explore the different concepts of the solution of an actual working environment. It is important to note that you, as an architect, have a heavy influence in which POC (Proof of Concept) will be used as a final solution, but the customer at end is the one who chooses which constraints and advantages suit their business better.

With the example of choosing an NGINX as a load balancer to provide high availability and performance improvements to APACHE web servers hosting the application files, we can implement a working solution with scaled down resources. Instead of deploying four nodes for the final solution, we can deploy just two to demonstrate the load balancing features as well as provide a practical demonstration of high availability by purposely bringing one of them down.

Here's a diagram describing the previous example:



This does not require the full four node cluster that was envisioned during the design phase, as we're not testing full performance of the entire solution. For performance or load testing, this can be done by having less concurrent users provide a close to actual workload for the application. While having less users will never provide exact performance numbers for the full implementation, it delivers a good baseline with data that can later be extrapolated to provide an approximation of what the actual performance will be.

As an example for performance testing, instead of having 2,000 users load the application, we can have a quarter of the user base and half of the resources. This will considerably decrease the amount of resources needed, while providing enough data to be able to analyze the performance of the final solution at the same time.

Also, in the Information gathering stage, a document that has the different proofs of concept documented is a good idea, as it can serve as a starting point if the customer wants to construct a similar solution in the future.

Implementing the solution

Once the customer has selected the optimal route based on their business needs, we can start constructing our design. At this stage, you will be facing different obstacles, as implementing the POC in a Dev or QA environment might vary from production. Things that worked in QA or Development may now fail in Production, and different variables might be in place; all these things only arise at the implementation stage, and you need to be aware that worse case scenario, it might mean changing a great part of the initial design.

This stage requires hands-on work with the customer and customer's environment, so it is of utmost importance to ensure that changes you make won't affect the current production. Working with the customer is also important, because this will familiarize their IT team with the new solution; this way, when the sign off is done, they are familiar with it and its configuration.

The creation of an implementation guide is one of the most important parts at this stage, since it will document each step and every minor configuration done to the solution. It will also help in the future in case an issue appears and the support team requires to know how it was configured in order to be able to solve the problem.

Summary

Designing a solution requires different approaches. This chapter went through the basics of the design stages and why each of them matters.

The first stage goes through analyzing the problem the design aims to solve, while at the same time asking the right questions. This will help define the actual requirements and narrow the scope to the real business needs. Working with the initial problem statement will impose problems further down the road, making this stage extremely important, as it will avoid unnecessarily going back and forth.

Then, we consider the possible paths or solutions we can take to solve the already defined problem. With the right questions asked in the previous stage, we should be able to construct several options for the customer to select and later implement a proof of concept. Proofs of concept help both customers and architects understand how the solution will behave in an actual working environment. Normally, POCs are scaled down versions of the final solution, making implementation and testing more agile.

Lastly, the implementation stage deals with the actual configuration and hands-on of the project. Based on the findings during the POC, changes can be made to accommodate the specifics of each infrastructure. Documentation delivered through this stage will help align parties to ensure that the solution is implemented as expected.

Further reading

In subsequent chapters, we'll go through the process of creating solutions for specific problems. As these solutions will be implemented in Linux, we recommend reading "Fundamentals of Linux".

Defining GlusterFS Storage

Everyday, applications require faster storage that can sustain thousands of concurrent IO requests. GlusterFS is a highly scalable, redundant filesystem that can deliver high-performance input–output to many clients simultaneously. We will define the core concept of a cluster and then introduce how GlusterFS plays an important role.

In this chapter, we will cover the following topics:

- Understanding the core concept of a cluster
- The reason for choosing GlusterFS
- Explaining software-defined storage
- Exploring the differences between a file, object, and block storage
- Explaining the need for a high performance and highly available storage

Technical requirements

This chapter will focus on defining GlusterFS; the link project's home is at `https://github.com/gluster/glusterfs` or `https://www.gluster.org/`.

Additionally, the project's documentation can be found at `https://docs.gluster.org/en/latest/`.

What is a cluster?

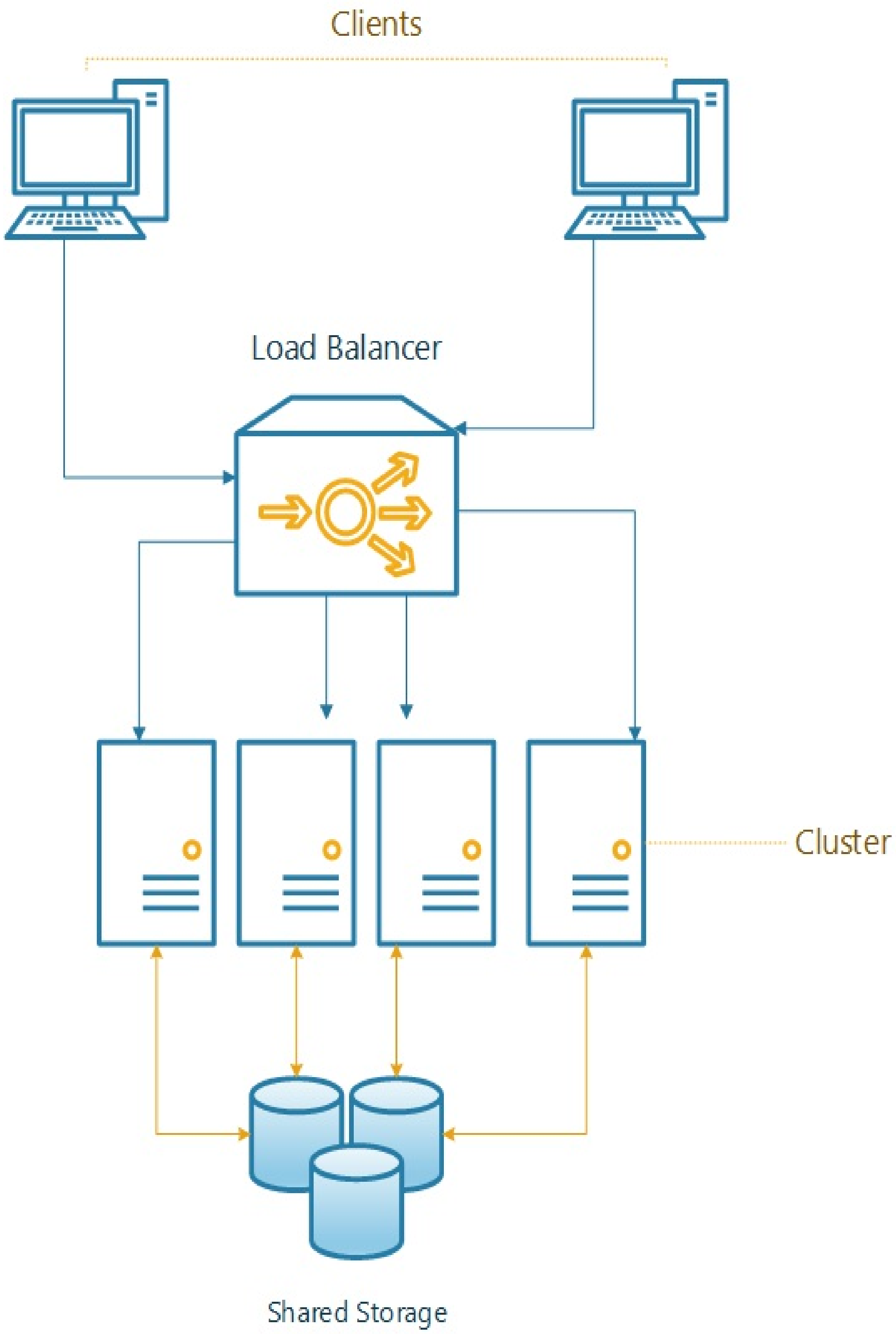
In the preceding chapter, we went through the different aspects of designing solutions to provide high availability and performance to applications that have high requirements. In the following chapters, we'll go through solving a very specific problem, that is, storage. For this, we can leverage the many advantages of a software-defined storage that allows for easy scalability and enhanced fault tolerance. GlusterFS is such a piece of software, which can create highly scalable storage clusters while providing maximum performance.

Before we go through how we can solve this specific need, we first need to define what a cluster is, why it exists, and what problems a cluster might be able to solve.

Computing a cluster

In simple words, a cluster is a set of computers, often called nodes, that work in tandem on the same workload that can distribute loads across all available members of the cluster to increase performance while at the same time allowing for self-healing and availability. Note that the term server wasn't used, as in reality any computer can be added to a cluster. From a simple Raspberry Pi to multi-CPU servers, clusters can be from a small two node configuration to thousands of nodes in a data center.

Here is an example of a Cluster:


Clients
Load Balancer
Cluster
Shared Storage

Technically speaking, clustering allows workloads to scale in performance by adding servers of the same kind with similar resource characteristics. Ideally, a cluster would have homogeneous hardware to avoid problems where nodes have different performance characteristics and at the same time make maintenance reasonably identical—this means hardware with the same CPU family, memory configuration, and software. The idea of adding nodes to a cluster allows to compute workloads to decrease their processing time. Depending on the application, compute times can sometimes decrease even linearly.

To further understand the concept of a cluster, imagine that you have an application that takes historical financial data. The application then receives this data and creates a forecast based on the stored information. On a single node, the forecast process (processes on a cluster are typically named jobs) takes roughly 6 days to complete, as we're dealing with several terabytes of data. Adding an extra node with the same characteristics decreases the processing time to 4 days. Adding a third node further decreases the time it takes to complete to 3 days.

Note that while we added three times the amount compute resources, compute time decreased only by about half. Some applications can scale performance linearly while others don't have the same scalability requiring more and more resources for fewer gains up to the point of diminishing returns. Adding more resources to obtain minimal time gain is not cost-effective.

From the above understanding, we can point out several characteristics that define a cluster:

- Help reduce processing time by adding compute resources

- Scale both vertically and horizontally
- Be redundant, that is, if one node fails, others should take the workload
- Allow for increased resources available for applications
- Being a single pool of resources rather than individual servers
- No single point of failure

Storage cluster

With a good understanding of what a cluster is, we can now move on to a storage cluster.

Instead of aggregating compute resources to decrease processing time, a storage cluster's main functionality is to aggregate available space to provide maximized space utilization while at the same time providing some form of redundancy. With the increased need for storing large amounts of data comes a need of being able to do that at a lowered cost while maintaining increased data availability. Storage clusters help solve this problem by allowing single monolithic storage nodes to work together as a large pool of storage space available. Thus, it allows storage solutions to reach the Petascale mark without the need to deploy specialized proprietary hardware.

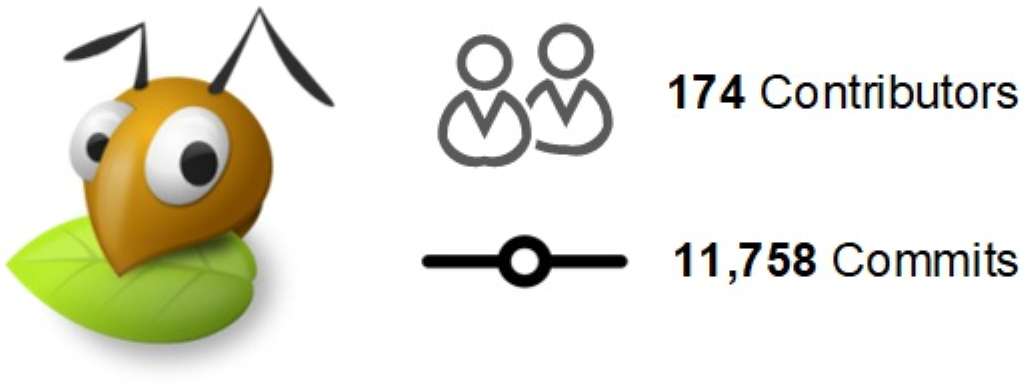
For example, consider that we have a single node with 500 TBs of available space and we need to achieve the 1PB (Petabyte) mark while providing redundancy. This individual node becomes a single point of failure because if it goes down, there's no way the data can be accessed. Additionally, we've reached the maximum hard disk drive (HDD) capacity available. In other words, we can't scale horizontally.

To solve this problem, we can add two more nodes with the same configuration, as the already existing one provides a total of 1 PB of available space. Now, let's do some math here, 500 TB times three should be approximately 1.5 PB, correct? The answer is most definitely yes. However, since we need to provide high availability to this solution, the third node acts as a "backup", making the solution tolerate a single node failure without interrupting the client's communication. This capability of allowing node failures is all thanks to the power of software-defined storage and storage clusters such as GlusterFS, which we'll explore next.

What is GlusterFS?

GlusterFS is an open source project by Gluster, which was acquired by Red Hat, Inc. in 2011. This acquisition does not mean that you have to acquire a Red Hat subscription or pay Red Hat to use it, since, as previously mentioned, it is an open source project; therefore, you can freely install it, see its source code, and even contribute yourself to the project. Red Hat offers solutions based on GlusterFS. However, we will talk about the Open Source Solution itself in this chapter.

The following is the number of contributors and commits on the gluster project:



To understand GlusterFS, we must understand how it differs from traditional storage, and to do this, we need to know the concepts behind software-defined storage (or SDS), which is what GlusterFS is.

Traditional storage is usually an industry standard storage array with proprietary software in it that is bound to the hardware vendor. All this restricts you to the following rules that your storage provider gives you:

1. Scalability limitations
2. Hardware compatibility limitations
3. Client-operating system limitations
4. Configuration limitations
5. Vendor lock-in

With software-defined storage, many, if not all, of these limitations are gone since it provides impressive scalability by not depending on any hardware. You can fundamentally take an industry standard server from any vendor that contains the storage you require and add it to your storage pool. By doing only this simple step, you are already overcoming four of the past limitations.

The previous example highly reduces the operating expense costs, or OPEX for short, as you do not have to buy additional high-priced expansion shelves for existing vendor storage array that can take weeks to arrive and be installed. You can quickly grab a server you have stored in the corner of your data center, and now it provides storage space for your existing applications. This process is called plug-in scalability and is present in most of the open source software-defined storage projects out there. In theory, by definition, the sky is the limit when it comes to scalability with SDS.

Software-defined storage scales when you add new servers to your storage pools and also increases the resilience of your storage cluster. Depending on what configuration you have, data is spread across multiple member nodes providing additional high availability by mirroring or creating parity for your data.

You also need to understand that software defined-storage is not creating space out of nothingness or separating what storage means from the hardware such as hard drives, solid-state drives, or any hardware device that is designed to store information. These will always be where that actual data is stored. SDS adds a logical layer that allows you to control where and how you store this data. It leverages this by its most fundamental components, an application programming interface or API that allows you to manage and maintain your storage cluster and logical volumes that provides the storage capacity to your other servers and applications and even monitoring agents that self-heal the cluster in case of degradation.

Software defined-storage is the future, and this is where the storage industry is moving, much so that it is predicted that in the next few years, approximately 70% of all current storage arrays will be available as software-only solutions or Virtual Storage Appliances (VSAs). Traditional NAS (network-attached storage) solutions are 30% more expensive than current SDS implementations, and mid-range disk arrays are even more costly. Taking all this into account with the fact that data consumption is growing approximately 40% in enterprise every year with a cost decline of only 25%, we can see why we are moving toward a software-defined storage world in the near future.

With the amount of applications running in public, private, or hybrid clouds, consumer and business data consumption is growing exponentially and ceaseless. This data is usually mission-critical and requires a high level of resiliency. The following are some of these applications:

- e-commerce and online storefronts
- Financial applications

- Enterprise resource planning
- Health care
- Big data
- Customer relationship management

When companies store this type of data called Bulk Data, they not only need to archive it but also need to access it and with the lowest latency possible. Imagine a scenario where you are sent to take x-rays during your doctor’s appointment, and when you arrive, they tell you that you have to wait for a week to get your scans because right now they have no storage space available to save your images. Naturally, this scenario will not happen because every hospital has a highly efficient procurement process, where they can predict usage based on their storage consumption and decide when to start the purchase and installation of new hardware, but you get the idea. It is way faster and efficient to install a POSIX-standard server into your software-defined storage layer and be ready to go.

Many other companies also require "data lakes" as secondary storage, mainly to store data in its raw form for analysis, real-time analytics, machine learning, and so on. Software Defined Storage is excellent for this type of storage, mainly because the maintenance required for this type of storage is minimum and also for the economic reasons we discussed previously.

We have been talking mainly about how economic and scalable SDS is, but it is also important to mention the high flexibility it brings to the table. SDS can be used for everything from archiving data and storing reach media to providing storage for Virtual Machines, as an endpoint for object storage in your private cloud and even in containers. It can be deployed on any of the previously mentioned infrastructures. It can run on your public cloud of choice, in your current on-premise virtual infrastructure, and even in a docker container or Kubernetes pod. In retrospective, it's so flexible that you can even integrate Kubernetes with GlusterFS using a RESTful management interface called "heketi" that dynamically provisions volumes every time you require persistent volumes for your Pods.

Block, File, and Object Storage

Now that we have gone through why software-defined storage is so amazing and the future for the next-generation workloads, it is time to dig a little deeper on the types of storage that we can achieve using SDS.

Traditional SAN and NAS solutions more commonly serve storage using protocols such as iSCSI, FC, FCoE and NFS, and SMB/CIFS. However, because we are moving more toward the Cloud, our storage needs change and this is where object storage comes in. We will explore a little bit more in depth about what Object Storage is and how does it compare to Block/File Storage, as GlusterFS is a File storage solution, but it also has Block and Object and storage capabilities.

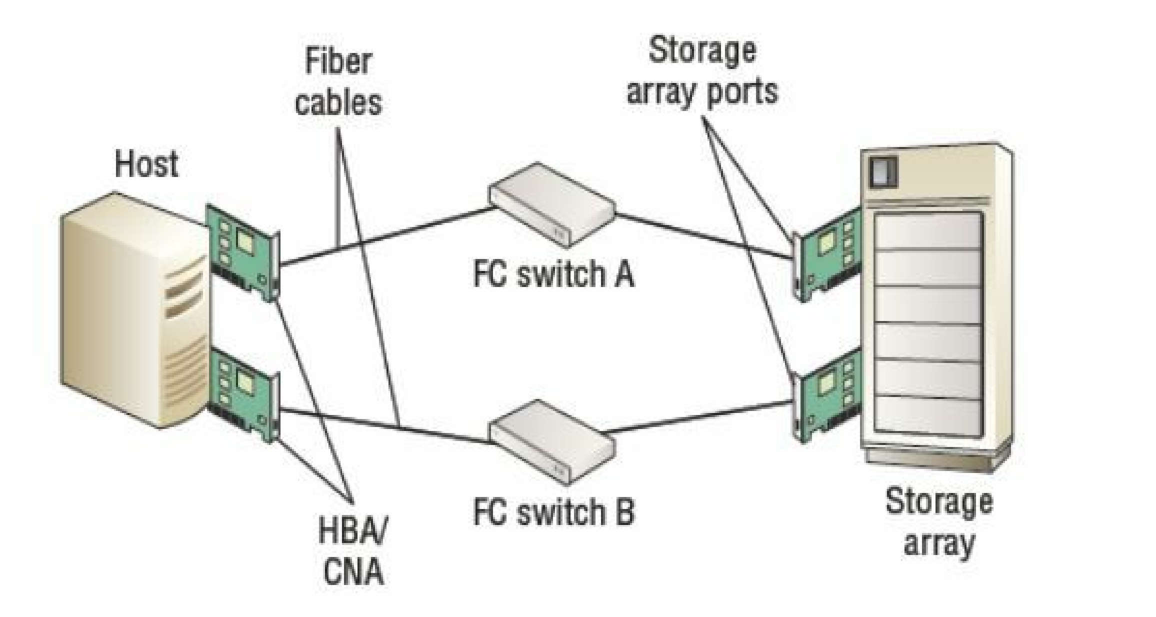
Block, File and Object:

Block storage, File storage, and Object storage work in very different ways when it comes to how the client stores its data in them, which causes their use cases to be completely different.

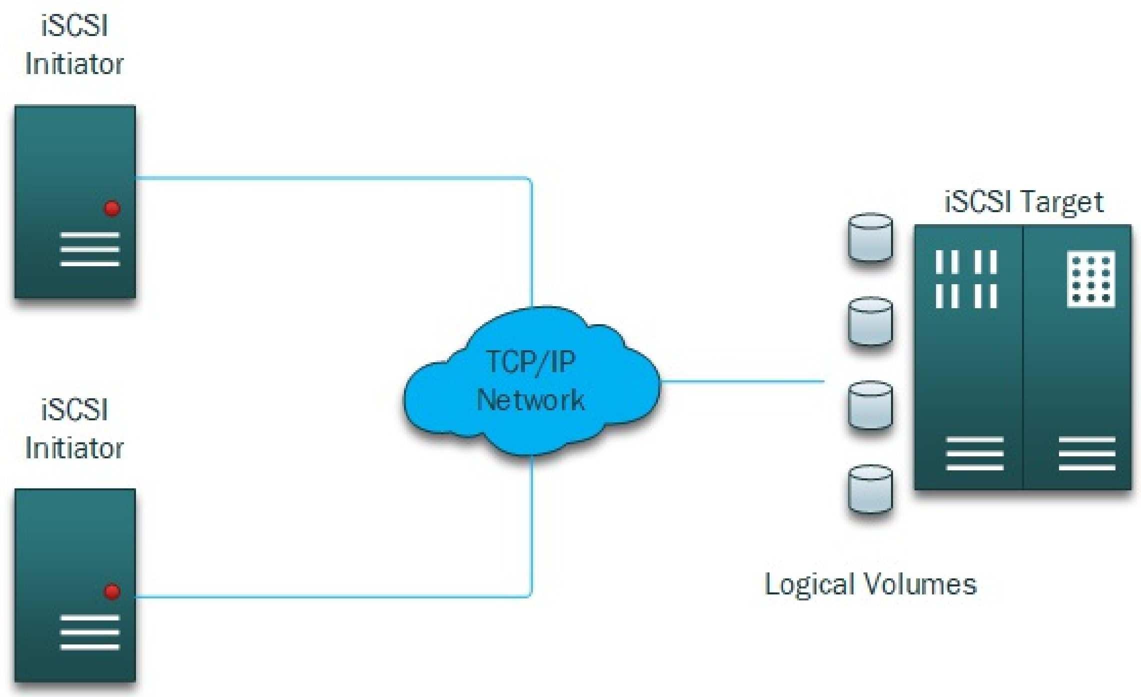
Block Storage

A SAN or Storage Area Network is where usually Block Storage is mainly utilized, using protocols such as FC (Fibre Channel) or iSCSI (Internet Small Computer Systems Interface), which are essentially mappings of the SCSI protocol over Fibre Channel and TCP/IP, respectively.

A typical FC SAN looks like this:



A typical iSCSI SAN looks like this:



Data is stored in logical block addresses, where when retrieving data the application usually says "I want the X amount of blocks from address XXYYZZZZ". This process tends to be very fast almost in the range of less of a millisecond, making this type of storage very low on latency and making it a very transactional-oriented type of storage form, making it ideal for random access. However, it also has its disadvantages when it comes to sharing across multiple systems, as block storage usually presents itself in its raw form, and you require a filesystem on top of it that can support multiple writes across different systems without corruption, in other words, a clustered filesystem.

This type of storage also has some downsides when it comes to high availability or disaster recovery, as we mentioned, it is presented in its raw form; therefore, the storage controllers/managers are not aware of how this storage is being used. When it comes to replicate its data to a recovery point, it only takes blocks into account, and some filesystems are terrible in reclaiming or zeroing blocks, which leads to unused blocks being replicated as well leading to deficient storage utilization.

Because of its advantages and low latency, Block Storage is perfect for structured databases, random read/write operations and also to store multiple Virtual Machines images which query the disks with hundreds if not thousands of I/O requests. For this clustered filesystem are designed to support multiple reads and writes from different hosts.

However, due to its advantages and disadvantages Block Storage requires quite a lot of "care and feeding", you need to take care not only of the filesystem and partitioning that you are going to put on top of your block devices. Additionally, you have to make sure that filesystem is kept consistent, secure, with correct permissions and without corruption across all the systems accessing it. Virtual Machines (VM) they have other filesystems stored in their virtual disks which also adds another layer of complexity into the picture, data can be written to the VM's filesystem and into the hypervisor's filesystem. Both filesystems have files that come and go, which need to be adequately zeroed for blocks to be reclaimed in a thin provisioned and replication scenario, and as we mentioned before, most storage arrays are not aware of the actual data being written to them.

File Storage

On the other hand, file storage or NAS (Network Attached Storage) is way more straightforward. You don’t have to worry about partitioning or about selecting and formatting a filesystem that suits your multi-host environment.

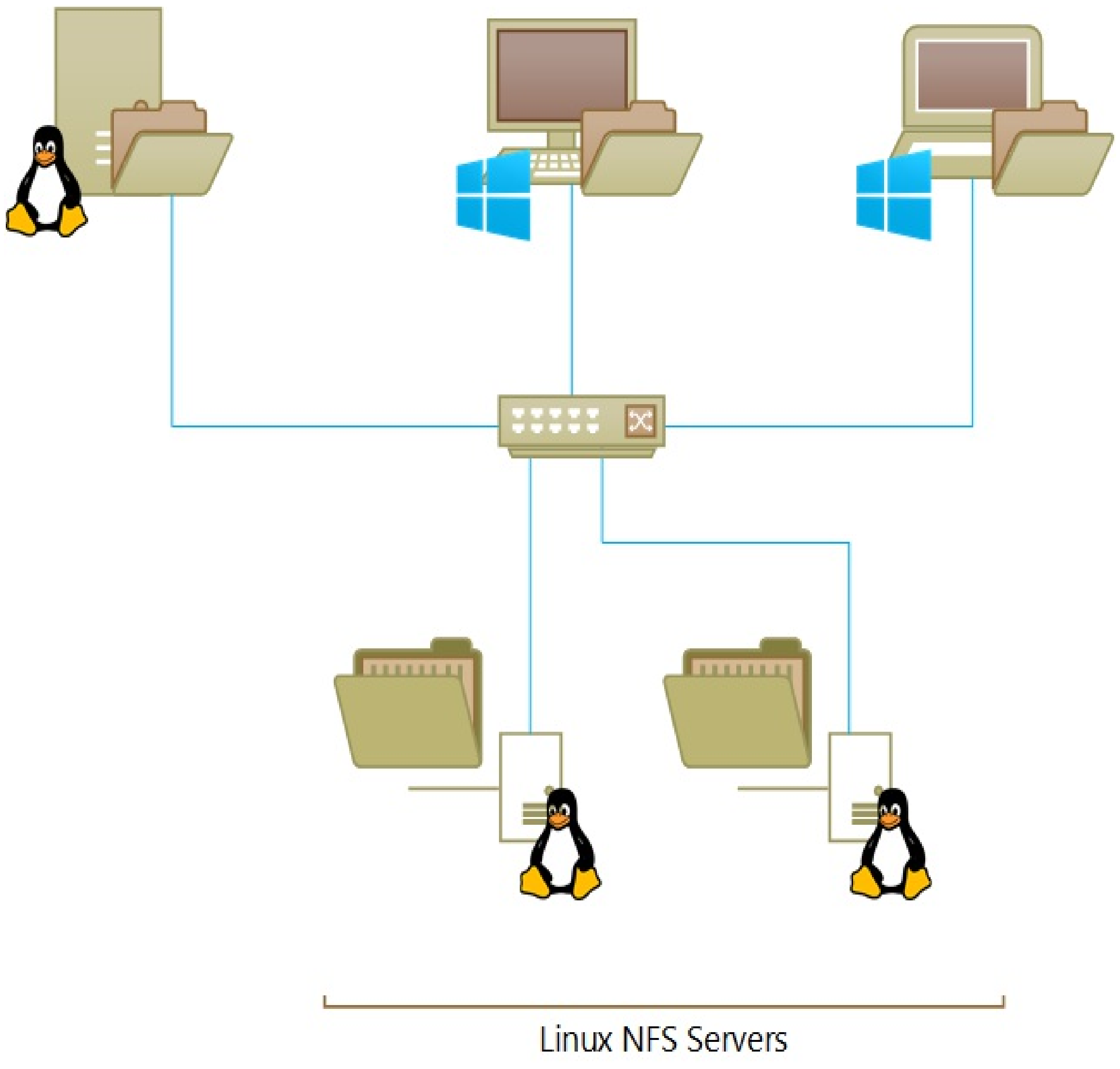
NAS are usually NFS or SMB/CIFS protocols, which are mainly used for storing data in "shared folders", unstructured data. These protocols are not very good at scaling or meeting the high media demands that we are facing in the cloud nowadays, such as social media serving and creating/uploading thousands of images or videos every day. This is where object storage saves the day, but we will be covering object storage later in this chapter.

File Storage, as the name says, works at the file level of the storage when you perform a request to a NAS; you are requesting a file or a piece of a file from the filesystem, not a bunch of logical addresses. With NAS, this process is abstracted to the client, and your storage array or your software-defined storage is in charge of accessing the disks on the backend and retrieving the file that you are requesting. File Storage also comes with its native features, such as file-locking, User and Group integration (when we are talking about Open Source Software, we are talking about NFS mainly here), Security, and Encryption.

Even though NAS storage abstracts and makes things simple for the client, it also has its downsides, as NAS relies heavily, if not entirely, on the network. It also has an additional filesystem layer with way higher latency than block storage. Many factors can cause latency or increase RTT (Round Trip Time). You have to take into consideration how many hops your NAS is away from your clients, the TCP window scaling, no Jumbo Frames on devices accessing your file shares, among others. Also, all these factors not only affect latency but are key players when it comes to the throughput of your NAS solution, which is where File Storage excels the most.

The following figure represents how versatile file storage share is:

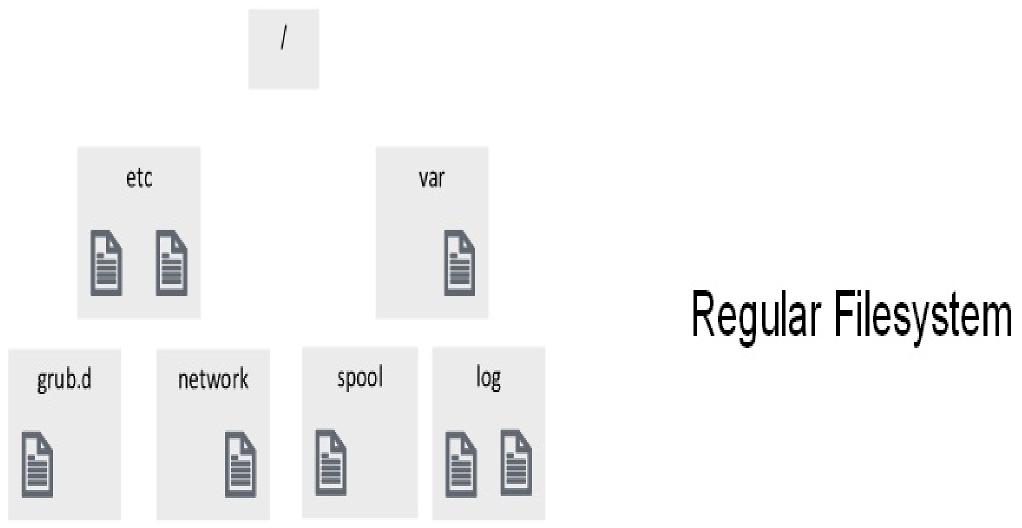
Different OS Clients Accessing the NFS Export

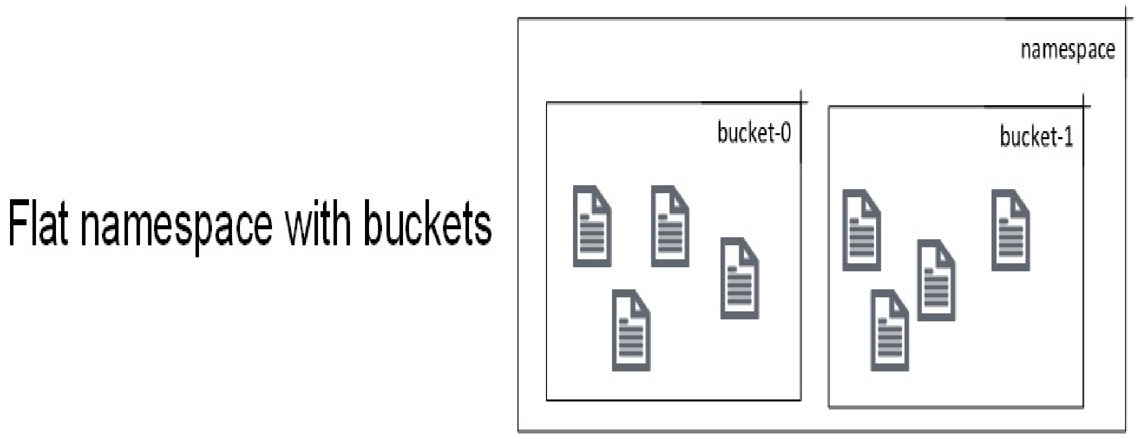


Object Storage

Object Storage is entirely different from NAS (file) and SAN (block). Although data is still being accessed through the network, the actual way data is retrieved is uniquely different. You will not be accessing files through a filesystem, but it will be via RESTful APIs using HTTP methods.

Objects will be stored in a flat namespace, which can store millions or billions of them; this is key to its high scalability, as it is not restrained to by the number of nodes like in regular filesystems, that is, XFS, EXT4. It is important to know that the namespaces can have partitions often called buckets, but they cannot be nested as regular folders in a filesystem, as the namespace is flat.

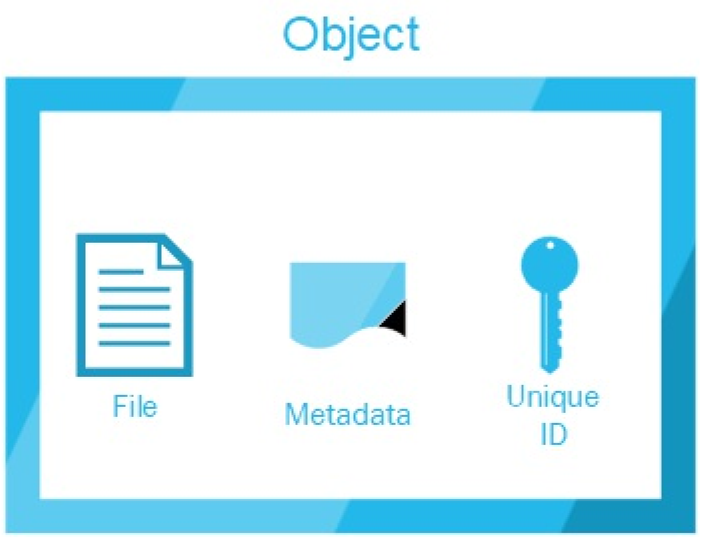




When comparing object storage with traditional storage, the self-parking versus valet parking analogy is often used. Why is this similar? Well, because in traditional filesystems when you store your file, you store it in a folder or directory, and it is your responsibility to know where that file was stored, just like parking a car in a parking spot—you need to remember the number and floor of where you left your car. With object storage, on the other hand, when you upload your data or put the file in a bucket, you are granted a unique identifier, which later you can use to retrieve it; you don't need to remember where it was stored. Just like a valet, who will go and get the car for you; you simply need to give them the ticket you received when you left your car.

Continuing with the valet parking reference, you usually give your valet more information about the car they need to get to you, not because they need it, but because they can identify your car better in this way—the color, plate number, or model of the car will help him a lot. With object storage, the process is the same. Each object has its own metadata, its unique ID, and the file itself are all part of the stored object.

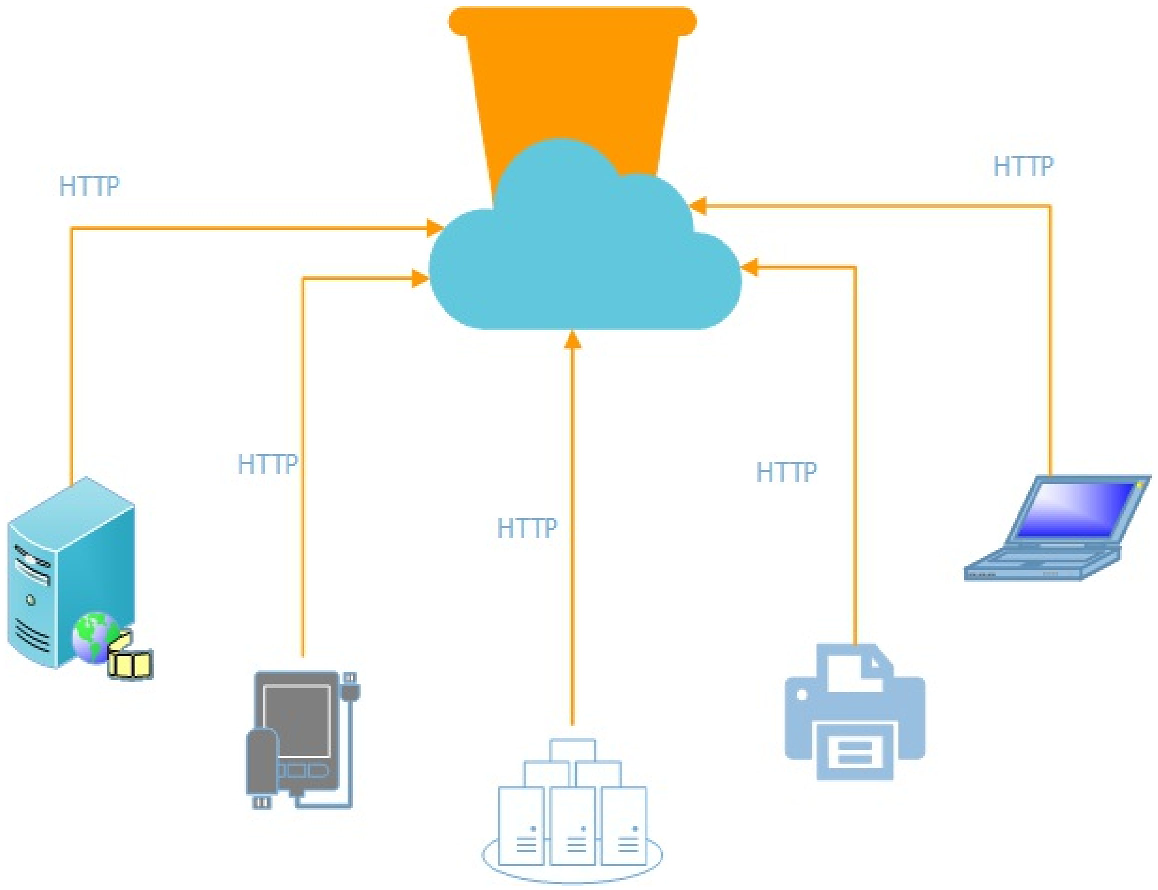
An object is object storage:



As we have mentioned several times, Object Storage is accessed through RESTful APIs. So, in good theory, any device that supports HTTP protocol can access your object storage buckets and PUT or GET any object. This sounds insecure, but, in fact, most of software-defined object storage have some type of authentication method, and you require an authentication token in order to be able to retrieve or upload files. A simple request using the Linux tool curl may look like this:

```
curl -X PUT -T "${path_to_file}" \
  -H "Host: ${bucket_name}.s3.amazonaws.com" \
  -H "Date: ${date}" \
  -H "Content-Type: ${contentType}" \
  -H "Authorization: AWS ${s3Key}:${signature}" \
  https://${bucket}.s3.amazonaws.com/${file}
```

Here, we can see how multiple distinct devices can connect to object storage buckets in the cloud via HTTP protocol:



Why choose GlusterFS?

Now that we understand the core concepts of Software Defined Storage, Storage Clusters, and the differences between Block, File, and Object storage, we can go through some of the reasons why enterprise customers choose GlusterFS for their storage needs.

As previously stated, GlusterFS is SDS or software-defined storage, a layer that sits on top of traditional local storage mount points, allowing the aggregation of storage space between multiple nodes into a single storage entity or a storage cluster. GlusterFS can run from out of the shelf-commodity hardware to private/public or hybrid clouds. Although its primary usage is file storage (NAS), several plugins allow it to be used as a backend for block storage via gluster-block and object storage via the gluster-swift plugins.

Some of the main features that define GlusterFS are as follows:

- Commodity hardware
- Can be deployed in private/ public or hybrid cloud
- No single point of failure
- Scalability
- Asynchronous geo-replication
- Performance

- Self-healing
- Flexibility

GlusterFS features

Let's go through each one of these features to understand why GlusterFS is so attractive for enterprise customers.

Commodity hardware – GlusterFS runs pretty much on anything

From ARM on a Raspberry Pi to any variety of x86 hardware, gluster requires merely local storage used as a brick, which lays the foundation storage for the volumes. There is no need for dedicated hardware or specialized storage controllers.

In its most basic configuration, a single disk formatted as XFS can be used with a single node. While not the best configuration, it allows for further growth by adding more bricks or more nodes.

Can be deployed in private/ public or hybrid cloud

From a container image to a full VM dedicated to GlusterFS, one of the main appeals for cloud customers is the fact that since GlusterFS is merely software, it can be deployed on private/ public or hybrid clouds. Since there is no vendor, locking volumes that span different cloud providers is entirely possible. Allowing for multi-cloud provider volumes with high availability setups is done so that when one cloud provider has problems the volume traffic can be moved to an entirely different provider with minimal-to-no downtime, depending on the configuration.

No single point of failure

Depending on the volume configuration, data is distributed across multiple nodes in the cluster, removing a single point of failure, as no "head" or "master" node controls the cluster.

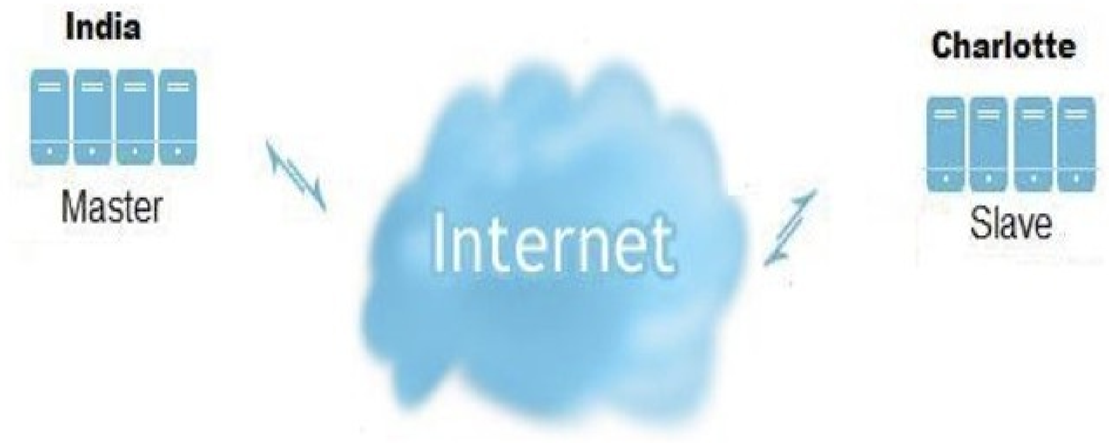
Scalability

GlusterFS allows for smooth scaling of resources by vertically adding new bricks or horizontally by adding new nodes to the cluster.

All this can be done online while the cluster serves data, without any disruption of the client's communication.

Asynchronous geo-replication

GlusterFS takes the "No single point of failure" concept that provides geo-replication, allowing data to be asynchronously replicated to clusters in entirely different geophysical data centers.



Performance

Since data is distributed across multiple nodes, we can also have multiple clients accessing the cluster at the same time. The process of accessing data from multiple sources at the same time is called parallelism, and GlusterFS allows for increased performance by directing the clients to different nodes. Additionally, performance can be increased by adding bricks or nodes, effectively by scaling horizontally or vertically.

Self-healing

In the case of unexpected downtime, the remaining nodes can still serve traffic. If new data is added to the cluster while one of the nodes is down, this data needs to be synchronised once the node is brought back up.

GlusterFS will automatically self-heal these new files once they're accessed, triggering a self-heal operation between the nodes and copying the missing data. This is transparent to the users and clients.

Flexibility

GlusterFS can be deployed on-premises on already existing hardware or existing virtual infrastructure, on the cloud as a Virtual Machine or as a container. There is no lock-in as to how it needs to be deployed, and customers can decide what fits their needs the best.

Remote Direct Memory Access

Remote Direct Memory Access (RDMA) allows for ultra-low latency extremely high-performance network communication between the gluster server and the gluster clients. GlusterFS can leverage RDMA for high-performance computing (HPC) applications and highly concurrent workloads.

Gluster Volume Types

Knowing the core features of GlusterFS, we can define the different types of volumes that GlusterFs provides. This will help in the next chapters as we dive into the actual design of a GlusterFS solution.

GlusterFS provides the flexibility of choosing the type of volume that best suits the workload needs, for example, for a high availability requirement, we can use a type of volume called replicated. This type of volume replicates the data between two or more nodes, having exact copies on each of the nodes.

Let's quickly list the types of volumes available, and later we'll discuss each of them and their advantages and disadvantages.

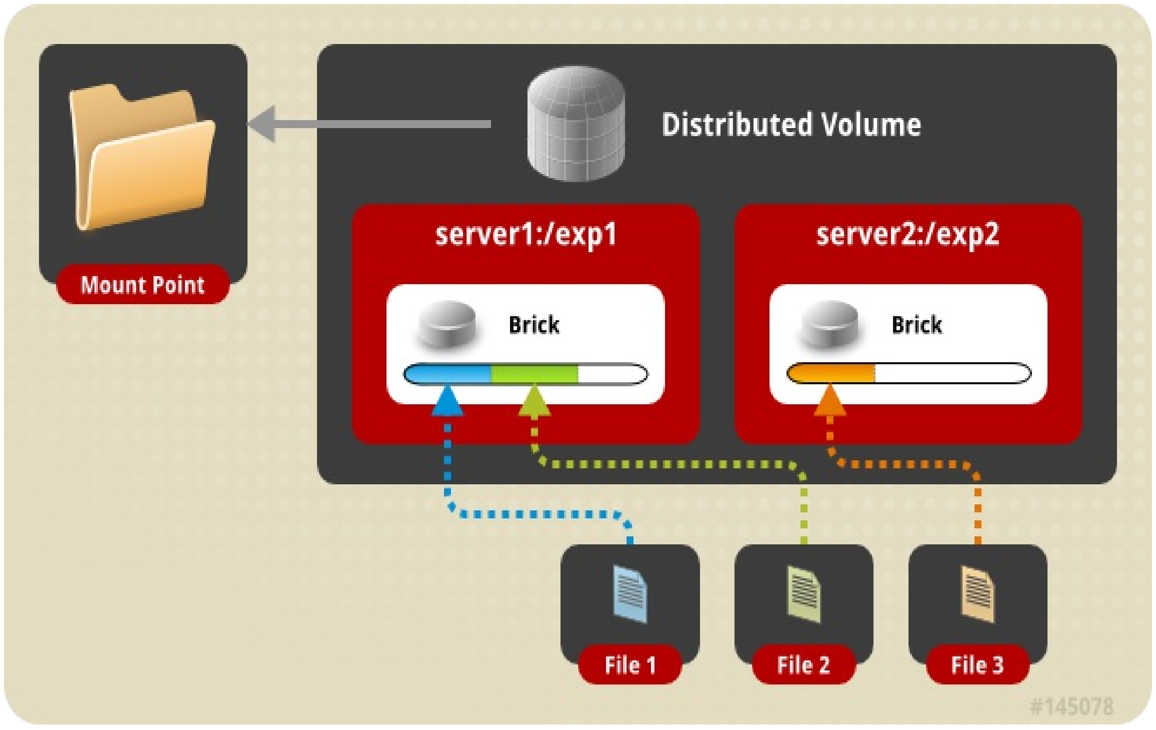
Here are some of the volume types:

- Distributed
- Replicated
- Distributed
- Replicated
- Dispersed
- Distributed dispersed

Distributed

As the name implies, data is distributed across the bricks in the volume and across the nodes. This type of volume allows for a seamless and low-cost increase in available space. The main drawback is that there is no data redundancy since files are allocated between bricks that could be on the same node or different nodes. It is mainly used for high storage capacity and concurrency applications.

Think of this volume type as a JBOD (Just a Bunch of Disks) or a linear LVM where space is just aggregated without any striping or parity.

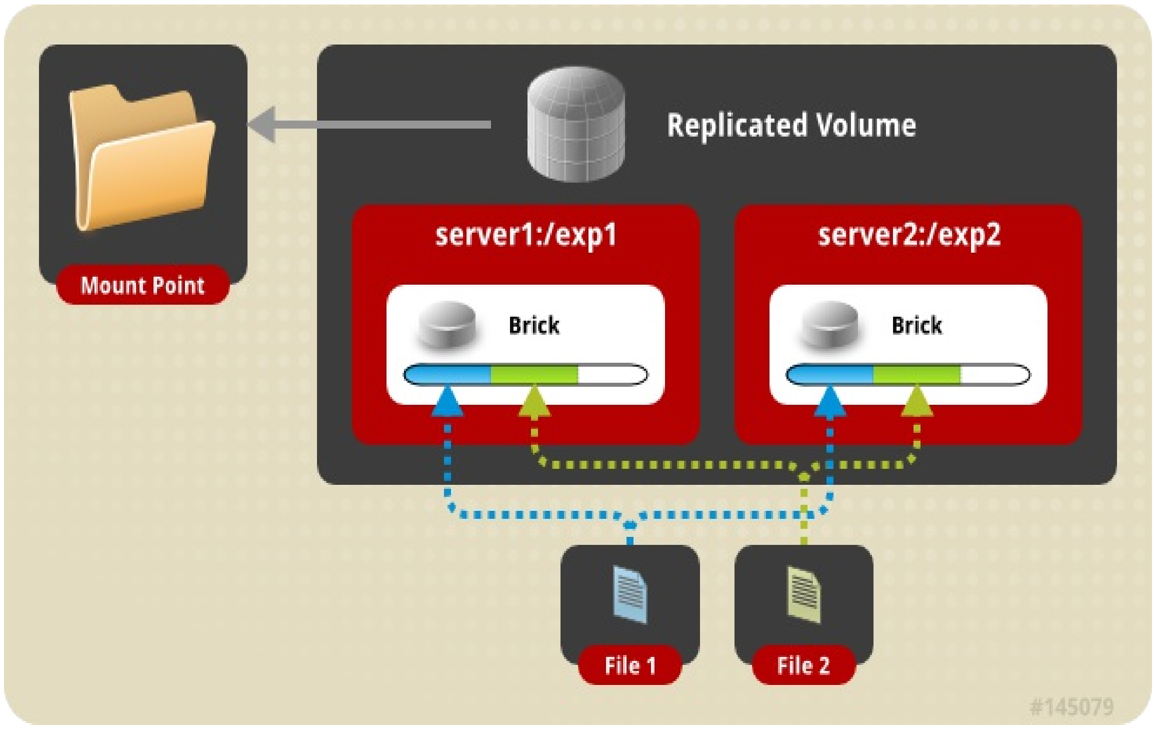


Replicated

In a replicated volume, data is copied across bricks on different nodes. Expanding a replicated volume requires the same amount of replicas to be added. For example, if I have a volume with two replicas and I want to expand it, I require a total of four replicas.

Compared to a RAID1 where data is mirrored between all available nodes. One of its shortcomings is that scalability is relatively limited. On the other hand, its main characteristic is high availability, as data is served even in the event of unexpected downtime.

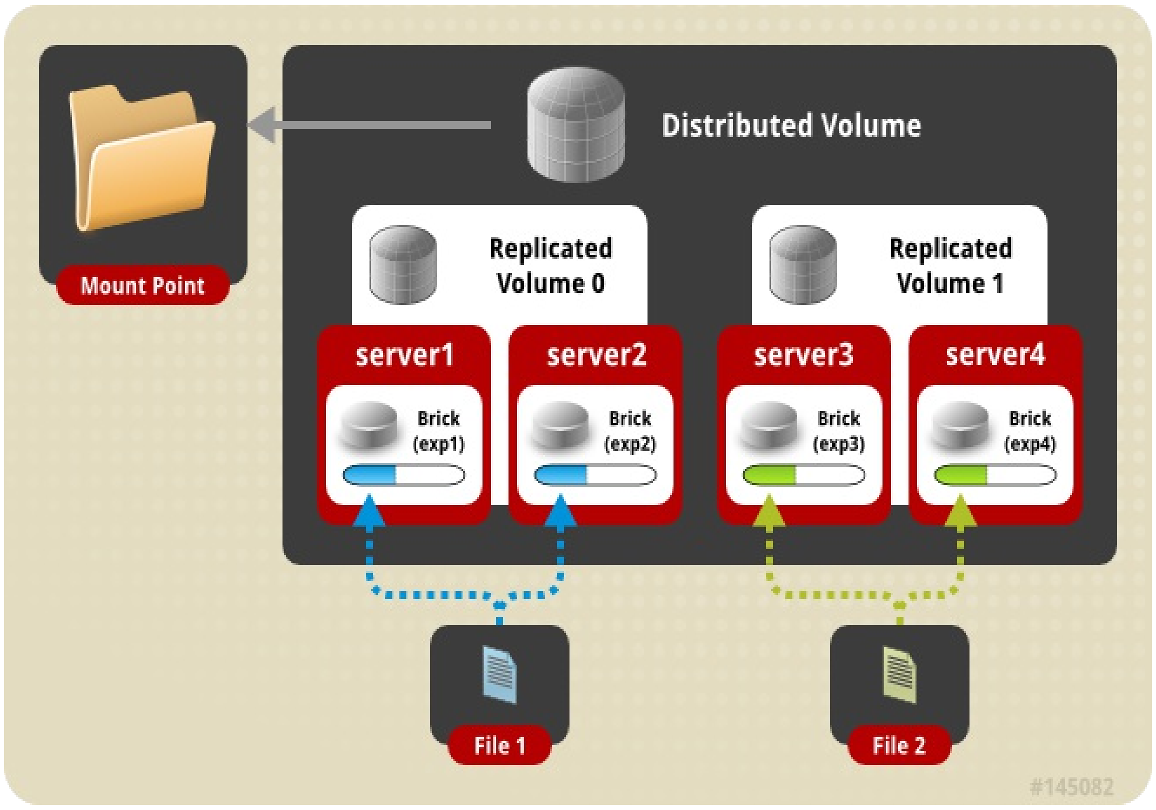
With this type of volume, mechanisms to avoid split-brain situations must be implemented. A split-brain occurs when new data is written to the volume, and different sets of nodes are allowed to process writes separately. Server Quorum is such a mechanism, as it allows for a tiebreaker to exist.


Replicated Volume
Mount Point
server1:/exp1
Brick
server2:/exp2
Brick
File 1
File 2
#145079

Distributed replicated

Distributed replicate is similar to a replicated volume, with the main difference that replicated volumes are distributed. To explain this, consider having two separate replicated volumes, each with 10 TB of space1. When both are distributed, the volume ends up with a total of 20 TB of space.

This type of volume is mainly used when both high availability and redundancy are needed, as the cluster can tolerate node failures.

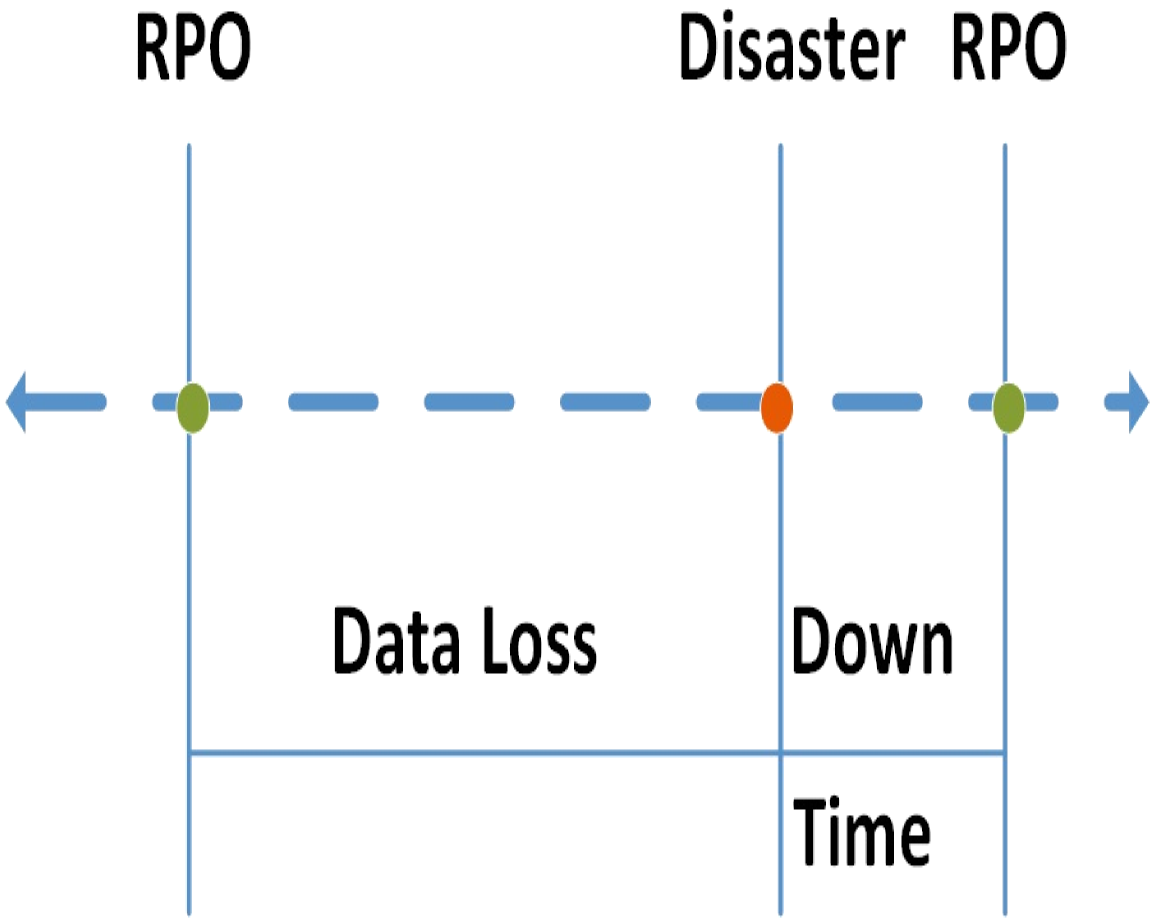


Disaster recovery

There's no escaping from it; disasters happen, whether its natural or human error. What counts is how well prepared we are for them and how fast and efficiently we can recover.

Implementing disaster recovery protocols is of utmost importance for business continuity. There are two terms that we need to understand before proceeding, and these are RTO or Recovery Time Objective and RPO (Recovery Point Objective). Let's take a quick glance. RTO is the time it takes to recover from a failure or event that causes a disruption, in simple words, how fast we can get the application back up. RPO, on the other hand, is how far the data can go back in time without affecting business continuity, that is, how much data I can lose.

The concept of RPO looks something like this:


RPO
Disaster
RPO
Data Loss
Down
Time

Dispersed

Dispersed volumes take the best of both distributed and replicated volumes by striping the data across all available bricks and at the same time allowing redundancy. Bricks should be of the same size, as the volume suspends all writes once the smallest brick becomes full. To further explain, imagine a dispersed volume such as a RAID 5 or 6, where data is stripped and parity is created, allowing data to be reconstructed from the parity. While the analogy helps to understand this type of volume, the actual process is entirely different as it uses erasure codes where data is broken into fragments. Dispersed volumes provide a right balance of performance, space, and high availability.

Distributed dispersed

In a distributed dispersed, volume data is distributed across volumes of type dispersed. Redundancy is provided at the dispersed volume level, having similar advantages of a distributed replicated volume.

Imagine a JBOD on top of two RAID 5 arrays—growing this type of volume requires an additional dispersed volume. While not necessarily the same size, ideally, it should maintain the same characteristics to avoid complications.

Need for highly redundant storage

With increased available space for applications comes an increased demand on the storage. Applications may require access to their information all of the time without any disruption that could cause the entire business continuity to be at risk. No company wants to have to deal with an outage, let alone an interruption in the central infrastructure that leads to money being lost, customers not being served, and users not being able to log in to their accounts all because of wrong decisions.

Let's consider storing data on a traditional monolithic storage array, doing this is prone to significant risks since everything is on a single place. A single massive storage array containing all of the company's information signifies an operational risk as the array is predisposed to fail. Every single type of hardware—no matter how good—fails at some point.

Monolithic arrays tend to handle failures by providing some form of redundancy through the use of traditional RAID methods done at the disk level. While this is good for small local storage that serves a couple of hundred users, this might not be a good idea when we reach the petascale as storage space and active concurrent users have increase drastically. In specific scenarios, a RAID recovery can cause the entire storage system to go down or degrade performance to the point that the application doesn't work as expected. Additionally, with increased disk sizes and single disk performance being the same for the past couple of years, recovering a single disk now takes a more substantial amount of time; it is not the same as rebuilding

1TB disks than 10TB disks.

Storage clusters, such as GlusterFS, handle redundancy differently by providing methods that fit the workload the best. For example, when using a replicated volume data is mirrored from one node to another, and in the case of a node going down, traffic is seamlessly directed to the remaining nodes, being utterly transparent to the users. Once the problematic node is serviced, it can be quickly put back into the cluster where it will go through a self-heal of the data. In comparison to traditional storage, a storage cluster removes the single point of failure by distributing data to multiple members of the clusters.

Having an increased availability means that we can reach the application service-level agreements maintaining the desired uptime.

RTO

As previously stated, this is the amount of time it takes to recover a functionality after a failure. Depending on the complexity of the solution, RTO might take a considerable amount of time.

Depending on the business requirements, RTO might be as low as a couple of hours. This is where designing a highly redundant solution comes into play by decreasing the amount of time required to be back operational.

RPO

This is the time data can go back which can be lost, in other words, this is how often recovery points are taken, in the case of backups, how often a backup is taken—it could be hourly, daily, or weekly—in the case of a storage cluster, how often changes are replicated.

One thing to take into consideration is the speed at which changes can be replicated, as we would like for changes to be replicated almost immediately, but due to bandwidth constraints, most of the times it is not possible.

Lastly, an essential factor to consider is how data is replicated. Generally, there are two types of replication, synchronous and asynchronous.

Synchronous replication

Synchronous means data is replicated immediately after it is written. This is useful to minimize RPO, as there is no wait or drift between the data from one node to another. A GlusterFS-replicated volume provides this kind of replication. Bandwidth should be considered, as changes need to be committed immediately.

Asynchronous replication

Asynchronous, on the other hand, means that the data is replicated in fragments of time, for example, every 10 minutes. During setup, the RPO is decided upon based on several factors, some being the business need and the available bandwidth.

Bandwidth is the primary consideration as depending on the size of the changes they might not fit in the RPO window, requiring a more considerable replication time directly affecting RPO times. If unlimited bandwidth is available, synchronous replication should be chosen.

In retrospective, we, as IT architects, spend a significant amount of time trying to figure out how to make our systems more resilient. Successfully decreasing RTO and RPO times marks the difference between a partial thought solution and an entirely architected design.

Need for high performance

With more and more users accessing the same resources, response times get slower, and applications start taking longer to process. Performance of traditional storage has not changed for the last couple of years, a single hard drive yields about 150MB/s with response times of several milliseconds. With the introduction of flash media and protocols such as NVMe (non-volatile memory express), a single Solid State Drive can easily achieve the gigabytes per second and sub-millisecond response times; Software Defined Storage can leverage these new technologies to provide increased performance and significantly reduce response times.

Enterprise storage is designed to handle multiple concurrent requests for hundreds of clients trying to get their data as fast as possible, but when the performance limits are reached, traditional monolithic storage starts slowing down causing applications to fail, as requests are not completed in time. Increasing performance of this type of storage comes at a higher price, and in most cases can't be done at all while the storage is still serving data.

The need for increased performance comes from the increased load in storage servers, with the explosion in data consumption users storing much more information and needing it much faster than before.

Applications also require data to be delivered to them as fast as possible, considering the stock market where data is requested multiple times a second by thousands of users. All this while, at the same time, another thousand users are continuously writing new data. If a single transaction is not committed in time, people will not be able to make the correct decision when buying or selling stocks because the wrong information was displayed.

The above problem is something that architects have to face when designing a solution that can deliver the expected performance necessary for the application to work as expected. Taking the right time to size storage solutions correctly makes the entire process flow smoother with less back and forth between design and implementation.

Storage systems, such as GlusterFS, can serve thousands of concurrent users simultaneously without a significant decrease in performance as data is spread across multiple nodes in the cluster. This approach is considerably better than accessing a single storage location like with traditional arrays.

Parallel Input Output (I/O)

Input-Output refers to the process of requesting and writing data to a storage system. The process is done through IO Streams where data is requested one block, file, or object at a time.

Parallel IO refers to the process where multiple streams are performing operations concurrently on the same storage system. This increases performance and reduces access times, as various files or blocks are read or written at the same time.

Contrary, serial IO is the process of performing a single stream of IO, which could lead to reduced performance and increased latency or access times.

Storage clusters such as GlusterFS take advantage of parallel IO since data is spread through multiple nodes, allowing for numerous clients to access data at the same time without any drop in latency or throughput.

Summary

In this chapter, we went through the core concepts of what a cluster is and defined that it is a set of computers called nodes working in together in the same type of workload. A compute cluster's primary function is to perform tasks that run CPU-intensive workloads, which are designed to reduce processing time. A storage cluster's function is to aggregate available storage resources into a single storage space that simplifies management and allows to efficiently reach the petascale or go beyond the 1PB available space.

Then, we spoke about how Software-Defined Storage is changing the way data is stored and how GlusterFS is one of the projects leading this change. SDS allows for simplified management of storage resources while at the same time adding features that were impossible with traditional monolithic storage arrays.

To further understand how applications interact with storage, we defined the core differences between Block, File, and Object storage. Mainly, block deals with logical blocks of data in a storage device, file works by reading or writing actual files from a storage space, and Object storage provides metadata to each object for further interaction.

With the concepts of the different interactions with storage, we went on pointing out the characteristics of GlusterFS that make it attractive for enterprise customers and how these features tie to what Software-Defined Storage stands for.

Finally, we delved into the main reasons why high availability and high performance are a must for every storage design and how performing parallel, or serial, Input-Output can affect application performance.

Up next, we will dive into the actual process of architecting a GlusterFS Storage Cluster.

Questions

1. How can I tune my storage?
2. What type of workloads glusterFS is better suited for?
3. Which cloud providers offer object based storage?
4. What’s the main difference between Ceph and Gluster?
5. What types of storage does Gluster offer?
6. Does Red Hat own Gluster?
7. Do I have to pay to use Gluster?
8. Does Gluster offer Disaster recovery or Replication?

Further Reading

1. Ceph Cookbook - Second Edition: Practical recipes to design, implement, operate, and manage Ceph storage systems by Vikhyat Umrao and Michael Hackett
2. Mastering Ceph: Redefine your storage system by Nick Fisk
3. Learning Ceph - Second Edition: Unified, scalable, and reliable open source storage solution by Anthony D'Atri and Vaibhav Bhembre

Architecting a storage cluster

Coming soon...

Using GlusterFS on cloud infrastructure

Coming soon…

Analyzing Performance in a Gluster system

Coming soon...

Creating a Highly available self-healing Architecture

Coming soon…

Understanding the core components of a Kubernetes cluster

Coming soon…

Architecting Kubernetes on Azure

Coming soon...

Deploying and configuring Kubernetes for your NGINX application

Coming soon…

Monitoring with ELK stack

Coming soon...

Designing an ELK Stack

Coming soon…

Using Elasticsearch, Logstash, and Kibana to Manage Logs

Coming soon...

Centralized Linux Patch Management with Spacewalk

Coming soon…

Lets take a Spacewalk

Coming soon…

Designing a simple but powerful Spacewalk solution

Coming soon...

Implementing a Spacewalk Infrastructure

Coming soon...

Design Best Practices

Coming soon...