UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA
(FINAL PROJECT PAPER)

IMPLEMENTACIJA DINAMIČNE DISTRIBUIRANE REŠITVE ZA SHRANJEVANJE PODATKOV Z UPORABO EDGEFS IN NAPRAV IOT (IMPLEMENTATION OF A DYNAMIC DISTRIBUTED DATA STORAGE SOLUTION USING EDGEFS AND IOT DEVICES)

MILAN NIKOLIĆ

ZAKLJUČNA NALOGA (FINAL PROJECT PAPER)

2021

NIKOLIĆ

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Implementacija dinamične distribuirane rešitve za shranjevanje podatkov z uporabo EdgeFs in naprav IoT

(Implementation of a dynamic distributed data
storage solution using EdgeFs and Iot devices)

Ime in priimek: Milan Nikolić
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Jernej Vičič
Somentor: asist. Aleksandar Tošić

Koper, september 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Milan NIKOLIĆ

Naslov zaključne naloge: Implementacija dinamične distribuirane reštive za sranjevanje podatkov z uporabo EdgeFs in naprav IoT

Kraj: Koper

Leto: 2021

Število listov: 43 Število slik: 27

Število referenc: 14

Mentor: izr. prof. dr. Jernej Vičič

Somentor: asist. Aleksandar Tošić

Ključne besede: EdgeFs, porazdeljeno shranjevanje, IoT naprave

Izvleček:
Zaključna projekta naloga predstavlja implementacijo rešitve za porazdeljeno shranjevanje podatkov z uporabo EdgeFs in naprav IoT. Bralcu je najprej predstavljen osnovni koncept porazdeljenega shranjevanja podatkov in primerjava s tradicionalno centralizirano hrambo. Opisane su različne rešitve za porazdeljeno hrambo podatkov in izbrana tehnologija EdgeFS. Ker obstajajo veliko možnosti za izbiro naprav IoT, so bralcu predstavljene najpogostejše naprave, naša izbira in obrazložitev zanjo. V zaključnem delu je bralcu predstavljeno izvajanje, preizkusi, ki so bili opravljeni, in njihovi rezultati.

Key words documentation

Name and SURNAME: Milan NIKOLIĆ

Title of final project paper: Implementation of a dynamic distributed data storage solution using EdgeFs and Iot devices

Place: Koper

Year: 2021

Number of pages: 43 Number of figures: 27

Number of references: 14

Mentor: Assoc. Prof. Jernej Vičič, PhD

Co-Mentor: Assist. Aleksandar Tošić

Keywords: EdgeFs, distributed storage, IoT devices

Abstract:
The final project presents the implementation of a distributed data storage solution using EdgeFs and IoT devices. The reader is first presented with the basic concept of distributed storage and a comparison of the traditional centralized storage implementations. We touch upon different distributed storage solutions and go into more detail with EdgeFs. The reader is presented with the most common device, our choice and the reasoning behind it, Since there are a lot of different options for an IoT device. In the end, the reader is presented with the implementation tests that were performed and their results.

Zahvala

Zahvalil bi se mentorju doc. dr. Jerneju Vičiču in somentorju mag. Aleksandru Tošiću za vso podporo, strokovno pomoč in usmeritve tako pri zaključnem delu, kot v času izobraževanja. Prav tako bi se zahvalil družini in prijateljem za vso podporo, ki so mi jo izkazali na izobraževalni poti.
Hvala!

Contents

1 Introduction **1**

2 Storage **4**
2.1 Centralized Storage 4
2.2 Distributed Storage 5
2.3 Distributed vs Centralized 6

3 IoT **8**
3.1 Most popular devices 9
3.2 Chosen IoT device: Raspberry Pi 11

4 Edge computing **13**
4.1 Cloud solutions 15
4.2 Docker 15
4.3 Kubernetes 16
4.4 Rook 17
4.5 EdgeFs 18
4.6 CephFs 19

5 Methodology and Implementation **21**
5.1 Setting up GCP and Kubernetes 21
5.1.1 Deploying EdgeFs 22
5.1.2 Deploying CephFs 24

6 Evaluation and Results **26**
6.1 EdgeFS 26
6.1.1 Test Description 26
6.1.2 Results 27
6.2 CephFs 28
6.2.1 Test Description 28
6.2.2 Results 29

7 Conclusion **31**

8 Povzetek naloge v slovenskem jeziku **32**

9 Bibliography **33**

List of Figures

1 A NeXTcube used by Tim Berners-Lee as the first Web server 1
2 WEB 1.0 vs WEB 2.0 2
3 WEB 1.0 vs WEB 2.0 3
4 Simplistic representation of the Web of Data 3

5 Logical representation of ZFS 5
6 Distributed Storage Example 5
7 Distributed vs Centralized model 6

8 Crude representation of an IoT system 8
9 Total number of device connections 10
10 Technical specification of Raspberry Pi 4 11
11 Raspberry Pie board 12

12 Representation of Edge architecture 13
13 May 2021 round-trip latency figures for Microsoft Azure 14
14 Multiple containers running on a Docker host 16
15 Application deployment 16
16 Kubernetes in relation to Docker 17
17 Rook Architecture 18
18 EdgeFs: Representation of data storage 19
19 Ceph Cluster Overview 19
20 Ceph Architecture Overview 20

21 Deployment representation 21
22 Console View of Kubernetes Cluster 22
23 Console View of disk configuration in one node 22
24 Cloudshell list of pods created by operator.yaml 23
25 Cloudshell list of pods created by cluster.yaml 23
26 Printout of FlexHash table 23
27 Cloudshell list of nodes created by cluster.yaml 24

List of abbreviations

CD	Compact Disk
HTML	HyperText Markup Language
RAID	Redundant Array of Inexpensive Disks or Redundant Array of Independent Disks
IoT	Internet of Things
IIoT	Industrial Internet of Things
IC	Integrated circuit
MIPI	Mobile Industry Processor Interface
DSI	Display Serial Interface
HDMI	High-Definition Multimedia Interface
DC	Direct current
USB-C	Universal Serial Bus Type C
GPIO	General-purpose input/output
PoE	Power over Ethernet
AWS	Amazon Web Services
SLA	Service-Level Agreement
OS	Operating System
VM	Virtual Machine

1 Introduction

In the early days of storage, sharing files was rather basic and rudimentary. In the early days of storage, sharing files was rather basic and rudimentary. Sharing data was done manually via floppy disks. Over the years technology continued developing and soon larger storage devices where available. But even with devices like CDs and hard disks which could store large quantities of data, nothing changed. Carrying an external device was still needed to physically connect to a computer. However, everything changed with the introduction of the Internet.
The Internet enabled a new way of sharing data, a new way of sharing information. We can now make connections to other computers all around the world.

Figure 1: A NeXTcube used by Tim Berners-Lee as the first Web server

Established in 1969 the Internet began as a small, publicly owned computer network [1]. The term 'Internet' emerged in 1974 as a simple abbreviation for internetworking between multiple computers [1]. Then in late 1989 Tim Burners-Lee introduced the World Wide Web [2]. The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI) [2]. Through its life cycle, the Word Wide Web underwent various phases of development transitioning through WEB 1.0 (Web of documents), WEB 2.0 (Web of people) and now slowly into WEB 3.0 (Web of data). Starting in 1989 and lasting until 2005 WEB 1.0 is defined as a web of information connections. This was the era of static pages with a purpose of only

[0]Source:https://en.wikipedia.org/wiki/Web_server

content delivery, allowing searching for information and reading. Technologies included in WEB 1.0 are HTLM, HTTP and URI.

A term coined by Tim O'Reilly Web 2.0 is a transition from the World Wide Web to a new phase of use and service development [3].

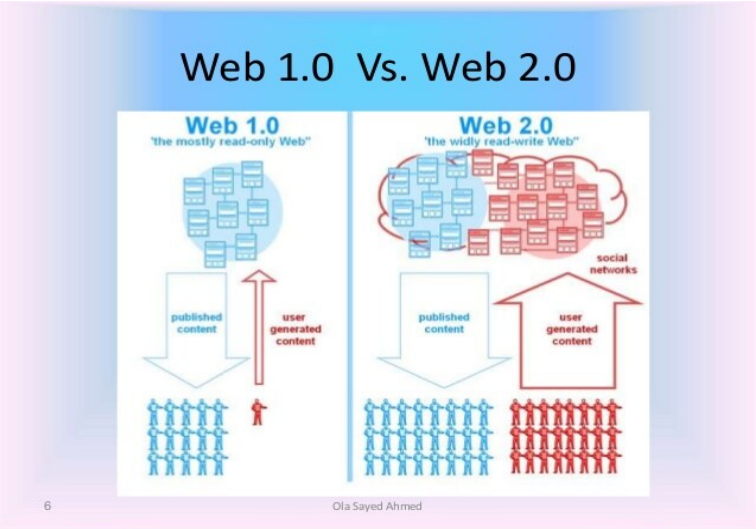


Figure 2: WEB 1.0 vs WEB 2.0

Sites that emphasize user-generated content, participation and interoperability (compatible to work with other systems and devices) are part of WEB 2.0. In the WEB 1.0 era users were limited to viewing content in a passive manner, whereas WEB 2.0 websites allow users to interact and collaborate with each other. An example of this would be social media sites like Facebook or Instagram, video sharing sites like YouTube, image sharing sites like Imgur, and many more. This is also recognized in this user centric definition [4] The Social Web is often used to characterize sites that consist of communities. It is all about content management and new ways of communication and interaction between users. Web applications that facilitate collective knowledge production, social networking and increases user to user information exchange [4].
When a user uses a site like Amazon to buy a certain item the website's algorithm will analyze and find other people who have purchased the same item, look at their other purchases and based on that will extrapolate and recommend items to the user. The website is learning and becoming more intelligent and this, in essence, is the very philosophy behind web 3.0.

[0]Source:https://www.slideshare.net/olaonyx/web-10-web-20-and-web-30/

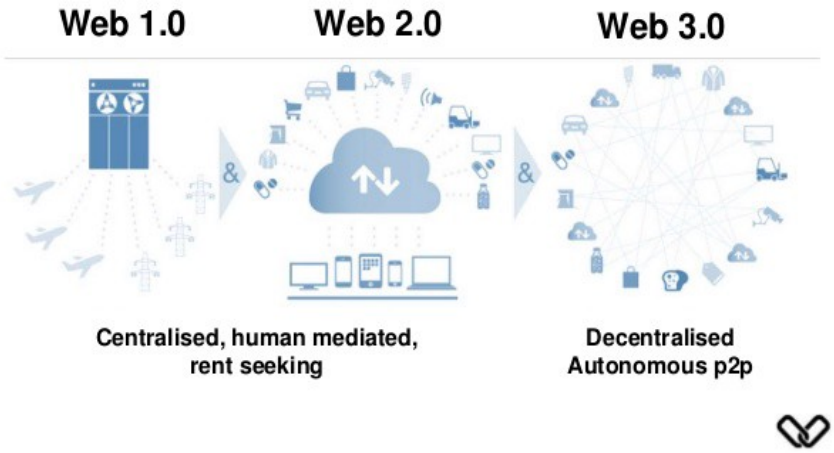


Figure 3: WEB 1.0 vs WEB 2.0

One of the key elements of WEB 3.0 is Semantic Web, a collaborative movement led by international standards body the World Wide Web Consortium. As defined on W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". It is a system that enables machines to "understand" and respond to complex human requests based on its meaning which requires for the information sources to be semantically structured. The Web of Data envisions data as being openly accessible to the general public hosting a variety of data sets like books, scholarly articles, metadata on music and many other types of information.

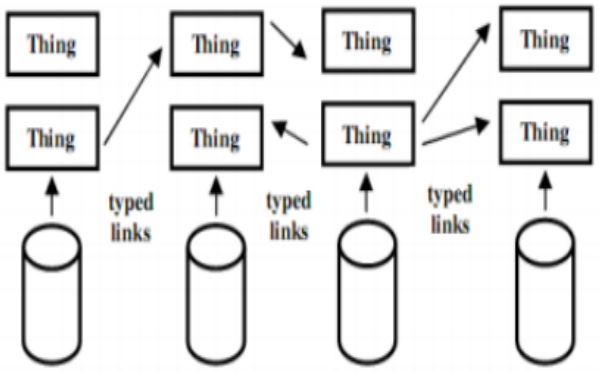


Figure 4: Simplistic representation of the Web of Data

[0]Source:https://boydcohen.medium.com/urban-mobility-web-2-0-uber-vs-web-3-0-iomob-2e424a99f8bd

[0]Source:https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.666.6445rep=rep1type=pdf

2 Storage

In a world heavily dependent on information one could argue that the way we store it is more important than the information itself. Most of our information today is stored on servers and most of those servers are owned by just a handful of large companies such as Microsoft, Google, Amazon, IBM. All of these servers are considered as centralized storage. But is this really the best way to store data, or should we explore a solution where data isn't stored on one big server but on many small ones scattered around the world?

2.1 Centralized Storage

In this era of WEB 2.0 data is stored in large data centers full of servers. Servers consist of physical disks combined together with a virtualization technology called "Redundant Array of Inexpensive Disks" (RAID) or a file system and logical volume manager like ZFS. The purpose of RAID is to achieve better data redundancy, performance improvement, or both. There are different RAID levels. All of them distribute data differently on the physical disks and offer a unique balance of reliability, availability, performance and capacity. Other than RAID 0, all levels provide protection against disk sector errors, as well as whole disk failures. In Case of ZFS, it controls how the bits and blocks of the file are stored and how drives are logically arranged for the purpose of RAID and redundancy. ZFS also allows snapshots, which can be described as photographs of how something was at a point in time. While traditional RAID requires the use of expensive hardware raid cards, ZFS does not. It also supports various RAID levels, some the same as with traditional RAID, namely RAID 0, 1, and 10, additionally it supports RAIDZ-1, RAIDZ-2 and RAIDZ-3. RAIDZ puts multiple drives together in a VDEV (logical grouping of one or more storage devices) and stores parity, or fault tolerance. Parity is stored across all drives in a VDEV with no dedicated parity drive. The amount of stored parity determines the RAIDZ level.

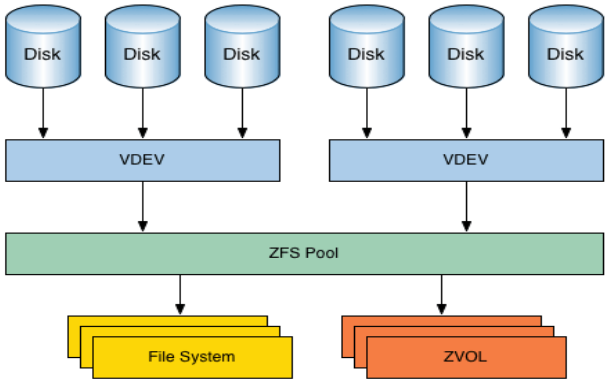


Figure 5: Logical representation of ZFS

The servers must be always operational. If a server hosting a web site is not workingthat web site cannot be accessed unless it is also hosted on another operational server.This high availability demand makes server maintenance very expensive and risk oflosing data always high.

2.2 Distributed Storage

Storing data has evolved during the years and now we are entering an era where the traditional approach to storage is no longer the optimal solution for both technical and economic reasons. We are approaching a point where just having faster drives and networks won't be enough. We need a new concept for storing data.

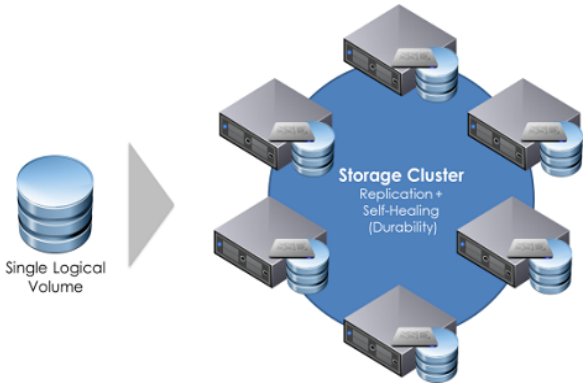


Figure 6: Distributed Storage Example

Rather than storing data in one centralized server, we use many servers that behave as one storage system even though data is distributed between these servers.

[0]Source:https://computingforgeeks.com/raid-vs-lvm-vs-zfs-comparison/

[0]Source:https://www.kdnuggets.com/2015/03/interview-dave-mccrory-basho-distributed-databases.html

2.3 Distributed vs Centralized

Looking at the statistic published by the "Internet World Stats" we can see that the number of Internet users over the last 10 years has gone up by more than 50%, from 2,267 million to 5,168 million [1]. TechJury states in one of their articles that in the year of 2020 people created 1.7MB of data every second and that by 2025 200+ zettabytes of data will be in cloud storage around the globe[2]. This further indicates that one of the most recent events in storage history is the transition from centralized to distributed storage. This is due to its ability to scale out capacity. Distributed storage is combined of many nodes, living in different locations. Each of those nodes can be used for both compute and storage, and scale up as more resources are needed.

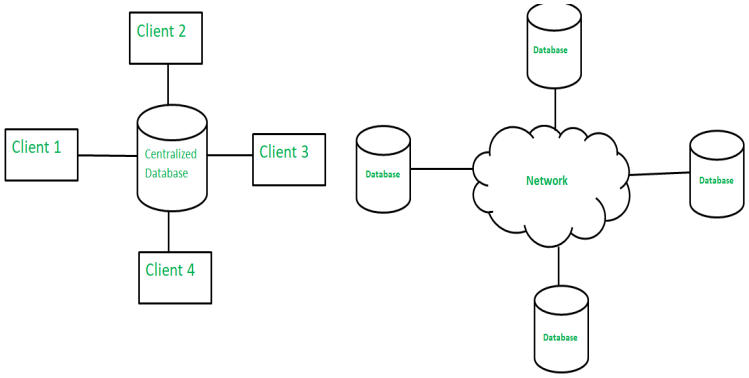


Figure 7: Distributed vs Centralized model

Both centralized and distributed storage have certain technical advantages and disadvantages. Centralized storage offers:

- Easier access and co-ordination of data, since it is stored in one location.
- For the same reason, minimal data redundancy.
- Lower cost compared to other data storage solutions for smaller use cases, since everything is stored, located and maintained in only one location.

However, some of the cons are:

- Problems when a large number of users try and access data at the same time, since one server has to handle all the requests.

[1]Source: https://www.internetworldstats.com/emarketing.htm
[2]Source: https://techjury.net/blog/how-much-data-is-created-every-day/
[0]Source: https://www.geeksforgeeks.org/difference-between-centralized-database-and-distributed-database/

- Higher risk of data loss- if any system failure occurs, all the data is lost.

Distributed storage offers:

- Easy expandability, since data is stored in different locations
- Easy access from different networks.
- Better security.
- Higher performance capability since system load is spread over multiple nodes.

On the other side, some of the disadvantages are:

- High cost and maintenance difficulty, due to higher complexity.
- No easy way to provide users with a uniform view, since data is not in one location.

Maybe more importantly distributed storage solves some key problems that are not technical, but rather ethical or even personal, like censorship and data control, since both are not possible in a distributed model.

3 IoT

The Internet of Things consists of physical objects, connected with other devices and systems over the Internet, that are embedded with sensors, software and other technologies for collecting and exchanging data. Internet of Things refers to the billions (probably more) devices connected to the Internet, all equipped with sensors for gathering information and an Internet connection for sending or receiving data, or even both. This opens up a whole new world of automation possibilities. IoT devices are capable of gathering information and then acting based on that information without any human input. For example: moisture sensor used in farming can tell if a field needs watering or the opposite - if it is too moist. That data can then be sent to the cloud where it is combined with other data from the Internet, like if rain is expected later in the day, and then decide if the field should be watered or not, thus keeping the ground at an optimal moisture level to yield a maximum profit from the field. With even more sensors this simple system can be turned into a much more complex system that can run algorithms that analyze all this information, leading to models that could be used to predict future conditions and prevent losses.

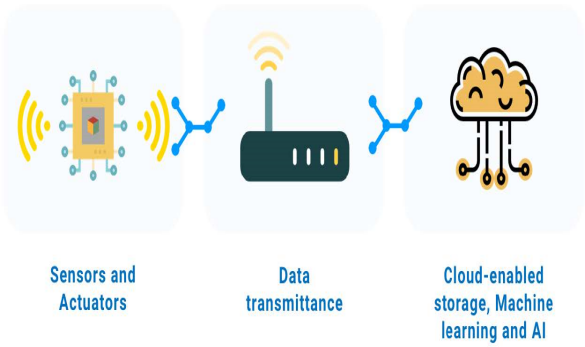


Figure 8: Crude representation of an IoT system

All IoT devices can be categorized and described as following:

- Devices that collect and send data.
- Devices that receive data and act on it.

[0]Source:https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

- Devices that can do both.

Combining sensors with an Internet connection gives devices that can collect and send data, thus allowing the collection of data about the environment, and make informed calculated decisions based on that information. For example, there is a street of 10 houses all connected to the electric grid via one base station which regulates the electricity for that street. A device can be installed to each house with sensors that measures how much electricity a house uses, sends that information to the cloud and uses algorithms to determine future patterns for electricity use to see when the most or least power is required. The base station in the example represents a device that can receive data and act on it. The electricity company can send a command to increase power to meet higher demand or decrease power if necessary. A more common example of such a device is a printer that could be as simple as in coming only with a "turn on" command, or a complex 3D model for a 3D printer [5]. The real power of the Internet of Things arises when things can do both of the above. Things that collect information and send it, but also receive information and act on it [5]. If in the above example every house is equipped with an IoT device that can collect, send, receive and act on information then the basis for a much more capable system is created. The IoT device can now collect data about the electricity use of a house, its temperature, humidity etc. It can then combine that data with data from the Internet such as weather predictions or electricity usage of other houses connected to the same network. By analyzing that data, it can decide and turn on or off the heating system or control any other device connected to it, resulting in a lowered electricity bill.
It is obvious that the power of IoT increases by increasing the number of sensors it has, devices it's connected to, devices it can control and by being backed by a powerful supercomputer in the cloud. But what are the benefits of IoT?
Beside the examples shown above IoT has many benefits for consumers and businesses. It makes the life of a consumer much easier, streamlining mundane everyday tasks, improving healthcare, lowering utility bills (as shown in the example above), and much more. Businesses can use IoT to increase efficiency, make informed decisions based on actual real-time data which gives an obvious competitive advantage, and much more.

3.1 Most popular devices

The number of IoT devices is growing fast and a large majority is made up of smartphones [6]. In fact, the total number of smartphone users topped 3 billion in 2018 and 7.9 billion in 2020 [6]. As shown in the image below (Figure 6), the number of devices is only expected to grow, and not just in number of smartphones, but also in the health-

care industry, construction, energy etc. [6]. The manufacturing and industrial sectors are early and fast adopters of IoT technology, a trend currently known as Industry 4.0. Industrial Internet of Things (IIoT) has a heavy focus not on people using machines, but on machine automation with minimal human input thanks to robotics [6].

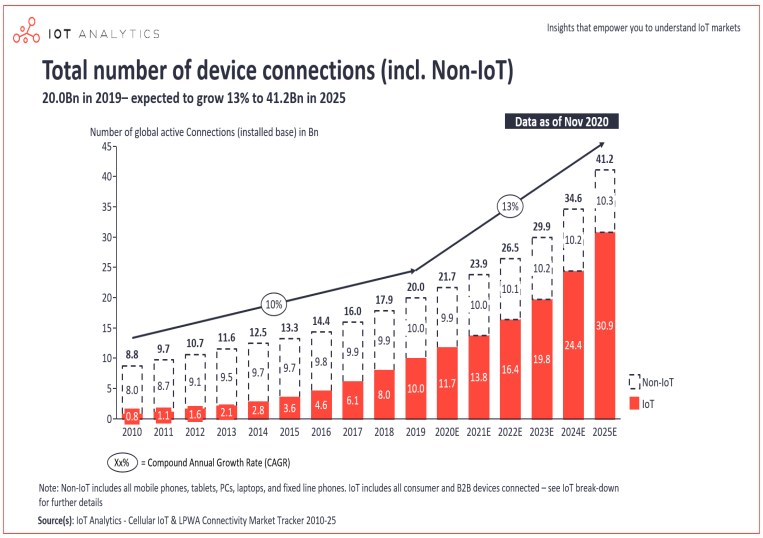


Figure 9: Total number of device connections

While the most popular IoT devices are smartphones and home automation devices, such as Google Home and Amazon Echo Plus we need to look at them from a more technical and developer focused point of view. At the heart of all consumer IoT devices lies a SoC - System on A Chip. For defining the term SoC we will use the definition from [8]. SoC design is defined as a complex IC that integrates the major functional elements of a complete end-product into a single chip or chipset. In general, SOC design incorporates a programmable processor, on-chip memory, and accelerating function units implemented in hardware. It also interfaces to peripheral devices and/or the real world. SOC designs encompass both hardware and software components [8]. In essence an IoT device is a SoC that includes hardware for Internet connection.
All the above mentioned IoT devices use a proprietary SoC that is not available to the general public, instead we will take a closer look at the IoT development board consumer market.
But first let's lay out the requirements for our development board

- Ability to run Linux, since it is our OS of choice.
- Ability to run Docker, for the ability to use containers

[0]Source:https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

- Ability to run Kubernetes, for automating container management.
- A minimum of 2GB of RAM, minimum specification for running Kubernetes as specified by Cloud66 [1].
- A Micro-SD card slot, for additional storage.

Unfortunately, our above requirements exclude all boards that are based on microcontrollers, since those IoT boards have a very small amount of RAM and storage available. While the huge IoT development board market offers many products suitable for our project, like the Odroid-XU4, Banana Pi M64, VIM2 SBC just to name a few, the Raspberry Pi will be used, due to its availability and huge online community.

3.2 Chosen IoT device: Raspberry Pi

Raspberry Pi is the obvious choice since it meets all the given requirements, it can run Linux, Docker and Kubernetes, up to a generous 8GB of RAM, and a Micro-SD card slot to install everything on.

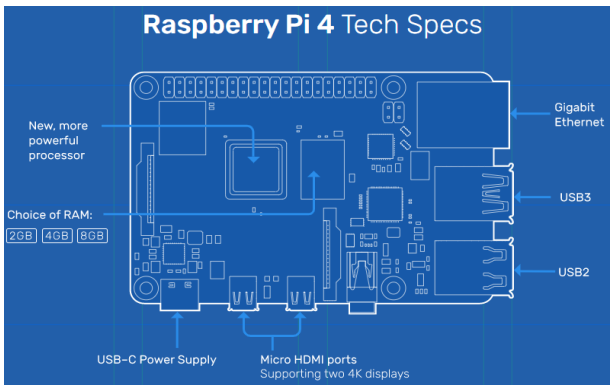


Figure 10: Technical specification of Raspberry Pi 4

The Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, should be powerful enough to ensure smooth operation of the board. It supports a 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless connection and even a Gigabit Ethernet port for wired connection, so connecting the Internet isn't a problem even in areas where wireless signal is weak or unstable. The 2-lane MIPI DSI display port and 2 micro-HDMI ports are welcome in case user interaction is required. The 40 pin GPIO header enables connecting a wide array of sensors for data gathering. It only needs 5V

[1]Source:https://help.cloud66.com/maestro/references/minimum-specs-kubernetes.html
[0]Source:https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

DC power to run and that can be via USB-C connector or GPIO header and it's even Power over Ethernet (PoE) enabled.

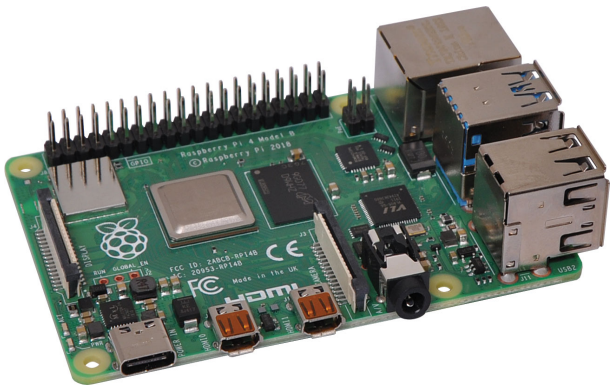


Figure 11: Raspberry Pie board

[0]Source:http://raspi.tv/2019/raspberry-pi-4-launches-with-bcm2711-quad-core-cortex-a72-64-bit-soc-running-at-1-5ghz-with-dual-4k-display-capability

4 Edge computing

Cloud based models depend on a data center to process data that is generated by edge devices. While that enables the use of large-scale analytics it also results in an increase in frequency of communication between an edge device and geographically distant cloud data centers. This is particularly a problem for services that need to process real-time data, like autonomous self-driving cars where a lot of data is generated every second from various sensors and sending that data to a centralized cloud and awaiting a response is not feasible due to bandwidth constraints and required quick response time. By offloading some of the processing from the data center to an edge device, chances of a network and data center overload are reduced, since we are reducing the number of device-cloud-device round trips, redundancy and availability are increased, since any possible disruption is limited to just one point in the network instead of the entire system, which is the case with the cloud. Data can be redirected through multiple pathways to the user ensuring availability.

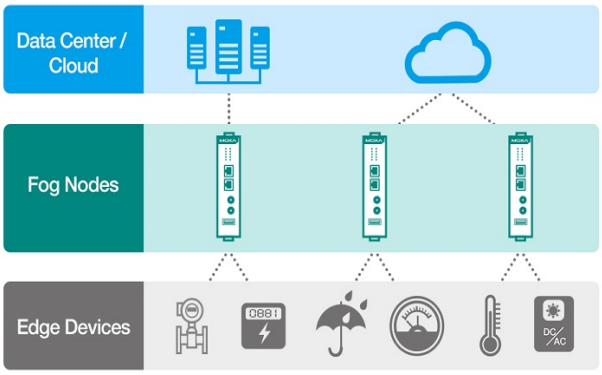


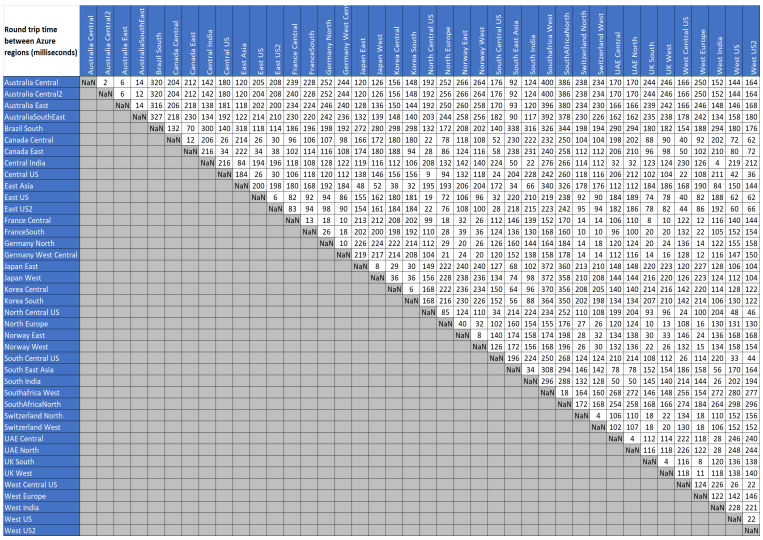
Figure 12: Representation of Edge architecture

In the figure above we have a representation of a better approach where computing and storage resources, known as cloudlets, micro datacenters or fog nodes, are placed near the end of the network closer to edge devices. Worth noting is that if the edge devices themselves have enough compute or storage resources they can be used as both an edge device and node, which is the case in our above example of autonomous self-driving cars.

[0]Source:https://www.moxa.com/en/articles/should-you-consider-fog-computing-for-your-iiot

[9] Edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services [9]. In this definition "Edge" refers to any computing and storage resource that is placed between the data center and the data source. Ideally as close to as possible to the data source, so that computing happens at the proximity of the data source. But why does this proximity matter?

Latency. Logical network proximity is almost entirely dependent on latency. When a large number of data streams are generated by edge devices real-time analytics is not possible when done by a distant cloud.



Round trip time between Azure regions (milliseconds)	Australia Central	Australia Central2	Australia East	AustraliaSouthEast	Brazil South	Canada Central	Canada East	Central India	Central US	East Asia	East US	East US2	France Central	FranceSouth	Germany North	Germany West Central	Japan East	Japan West	Korea Central	Korea South	North Central US	North Europe	Norway East	Norway West	South Central US	South East Asia	South India	SouthAfrica West	SouthAfricaNorth	Switzerland North	Switzerland West	UAE Central	UAE North	UK South	UK West	West Central US	West Europe	West India	West US	West US2
Australia Central	NaN	2	6	14	320	204	212	142	180	120	205	208	239	228	252	244	120	126	156	148	192	252	266	264	176	92	124	400	386	238	234	170	170	244	246	166	250	152	144	164
Australia Central2		NaN	6	12	320	204	212	142	180	120	204	208	240	228	252	244	120	126	156	148	192	256	266	264	176	92	124	400	386	238	234	170	170	244	246	166	250	152	144	164
Australia East			NaN	14	316	206	218	138	181	118	202	200	234	224	246	240	128	136	150	144	192	250	260	258	170	93	120	396	380	234	230	166	166	239	242	166	246	148	146	168
AustraliaSouthEast				NaN	327	218	230	134	192	122	214	210	230	220	242	236	132	139	148	140	203	244	258	256	182	90	117	392	378	230	226	162	162	235	238	178	242	134	158	180
Brazil South					NaN	132	70	300	140	318	118	114	186	196	198	192	272	280	298	298	132	172	208	202	140	338	316	326	344	198	194	290	294	180	182	154	188	294	180	176
Canada Central						NaN	12	206	26	214	26	30	96	106	107	98	166	172	180	180	22	78	118	108	52	230	222	232	250	104	104	198	202	88	90	40	92	202	72	62
Canada East							NaN	216	34	222	34	38	102	114	116	108	174	180	188	94	28	86	124	116	58	238	231	240	258	112	112	206	210	96	98	50	102	210	80	72
Central India								NaN	216	84	194	196	118	108	128	122	119	116	112	106	208	132	142	140	224	50	22	276	266	114	112	32	32	123	124	230	126	4	219	212
Central US									NaN	184	26	30	106	118	120	112	138	146	156	156	9	94	132	118	24	204	228	242	260	118	116	206	212	102	104	22	108	211	42	36
East Asia										NaN	200	198	180	168	192	184	48	52	38	32	195	193	206	204	172	34	66	340	326	178	176	112	112	184	186	168	190	84	150	144
East US											NaN	6	82	92	94	86	155	162	180	181	19	72	106	96	32	220	210	219	238	92	90	184	189	74	78	40	82	188	62	62
East US2												NaN	83	94	98	90	154	161	184	184	22	76	108	100	28	218	215	223	242	95	94	182	186	78	82	44	86	192	60	66
France Central													NaN	13	18	10	213	212	208	202	99	18	32	26	112	146	139	152	170	14	14	106	110	8	10	122	12	116	140	144
FranceSouth														NaN	26	18	202	200	198	192	110	28	39	36	124	136	130	168	160	10	10	96	100	20	20	132	22	105	152	154
Germany North															NaN	10	226	224	222	214	112	29	20	26	126	160	144	164	184	14	18	120	124	20	24	136	14	122	155	158
Germany West Central																NaN	219	217	214	208	104	21	24	20	120	152	138	158	178	14	14	112	116	14	16	128	12	116	147	150
Japan East																	NaN	8	29	30	149	222	240	240	127	68	102	372	360	213	210	148	148	220	223	120	227	128	106	104
Japan West																		NaN	36	36	156	228	238	236	134	74	98	372	358	210	208	144	144	216	220	126	223	124	112	104
Korea Central																			NaN	6	168	222	236	234	150	64	96	370	356	208	205	140	140	214	216	142	220	114	128	122
Korea South																				NaN	168	216	230	226	152	56	88	364	350	202	198	134	134	207	210	142	214	106	130	122
North Central US																					NaN	85	124	110	34	214	224	234	252	110	108	199	204	93	96	24	100	204	48	46
North Europe																						NaN	40	32	102	160	154	155	176	27	26	120	124	10	13	108	16	130	131	130
Norway East																							NaN	8	140	174	158	174	198	28	32	134	138	30	33	146	24	136	168	168
Norway West																								NaN	126	172	156	168	196	26	30	132	136	22	26	132	15	134	158	154
South Central US																									NaN	196	224	250	268	124	124	210	214	108	112	26	114	220	33	44
South East Asia																										NaN	34	308	294	146	142	78	78	152	154	186	158	56	170	164
South India																											NaN	296	288	132	128	50	50	145	140	214	144	26	202	194
SouthAfrica West																												NaN	18	164	160	268	272	146	148	256	154	272	280	277
SouthAfricaNorth																													NaN	172	168	254	258	168	166	274	184	264	298	296
Switzerland North																														NaN	4	106	110	18	22	134	18	110	152	156
Switzerland West																															NaN	102	107	18	20	130	18	106	152	152
UAE Central																																NaN	4	112	114	222	118	28	246	240
UAE North																																	NaN	116	118	226	122	28	248	244
UK South																																		NaN	4	116	8	120	136	138
UK West																																			NaN	118	11	118	138	140
West Central US																																				NaN	124	226	26	22
West Europe																																					NaN	122	142	146
West India																																						NaN	228	221
West US																																							NaN	22
West US2																																								NaN

Figure 13: May 2021 round-trip latency figures for Microsoft Azure

From the figure above we can see that the round-trip time form North Europe to South East Asia is 160ms. [10] For example, remote augmented reality applications which analyze a camera video feed to determine where to place a virtual object need a latency as low as 20-50ms [10]. So, if a user is in South East Asia and all the processing is done in North Europe than the round-trip time exceeds our latency requirement which may diminish the user experience. Having a cloudlet in proximity would result in a highly responsive cloud service, by lowering the end-to-end latency, increase scalability with edge analytics, by processing the raw data received from the source and sending only a small portion of it to the main cloud, help enforce privacy policies, by ensuring the privacy policies of the owner are fulfilled before sending the data to the cloud, and mask cloud outages, by temporarily acting as a fallback service if the cloud becomes unavailable.

[0]Source:https://docs.microsoft.com/en-us/azure/networking/azure-network-latency

With this project we will try and achieve all of the before-mentioned benefits of edge computing. By using an Raspberry Pi device as both an edge device and a cloudlet, we will try and connect multiple devices with each other to share data, which will hopefully increase data redundancy and prevent data loss. One device will be a 'master' device, which will periodically send analyzed and sorted data (significantly smaller than the original data) to a remote server. In case the master device brakes, an algorithm will choose another suitable master to replace it. Each device will be able to collect data from sensors connected to it, analyze and store the data, and then take action without any human input. Finally, we will do tests which will help us to analyze our latency, speed, reliability and data redundancy.

4.1 Cloud solutions

The idea of a decentralized edge computing system is, at this point in time, not a revolutionary idea by no means. There are many cloud services available that offer various ways to interact with and control remote devices. Cloud services like balena, Amazon Web Services (AWS), Microsoft Azure and Google Cloud all have IoT specific services [11]. Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs [11]. While cloud services do offer ways to control our remote devices, they are also built under the assumption that devices will be used only for data collection. The expectation is that devices just collect and send data back to the cloud, which then uses its processing power to do complex analytics and, when needed, send instructions back to devices. In our project we will rely on cloud services as only a final storage location for our data. But the main objective is to create a system where data is being analyzed and used closer to its origin.

4.2 Docker

To understand what Docker is we must first define what a container is. A container is an application or services, with all of its dependencies and configuration packaged together in an image. That container can then be tested and deployed as an image instance to the host operating system. All containers hosted on an OS are isolated and run on a container host which in turn runs on the OS. The largest benefit of this approach is scalability, by creating a new container for a short-term task.

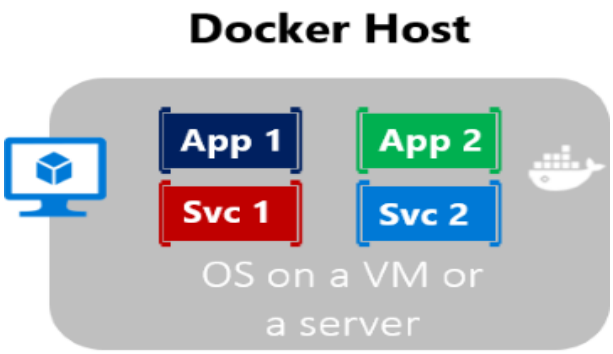


Figure 14: Multiple containers running on a Docker host

Docker is just an open-source engine that automates the deployment of containers. All docker image can run natively on Windows or Linux, but Windows images can only run on Windows, and Linux images only on Linux.

4.3 Kubernetes

To fully understand Kubernetes, we need to take a look at how we deploy applications.

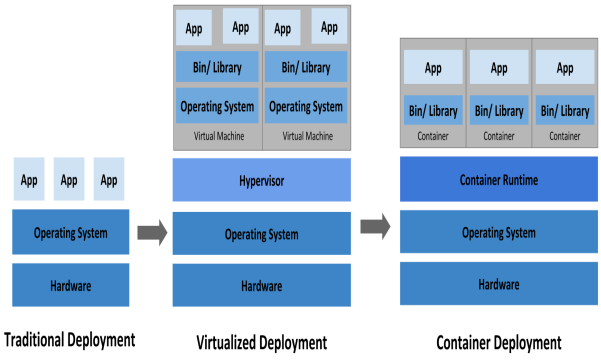


Figure 15: Application deployment

In the first column, following the traditional deployment, applications ran on physical servers, with no way of allocating resource and defining resource boundaries. So, if multiple applications run on one physical server, one application could take up most of the server's resources which would result in other applications under-performing. The solution for this is to run multiple virtual machines on one server, each running its own OS. Each application then would run in its own VM, with no access to other

[0]Source:https://docs.microsoft.com/ro-ro/dotnet/architecture/microservices/container-docker-introduction/

[0]Source:https://www.softwaredaily.com/topic/container-orchestration-wars

applications. The last column, container deployment, we use containers instead of VM, which are very similar, the major difference being that container share the OS. In a production environment, where we have one or multiple applications running in multiple containers, we need a way to manage them, and this is where Kubernetes comes in to play.

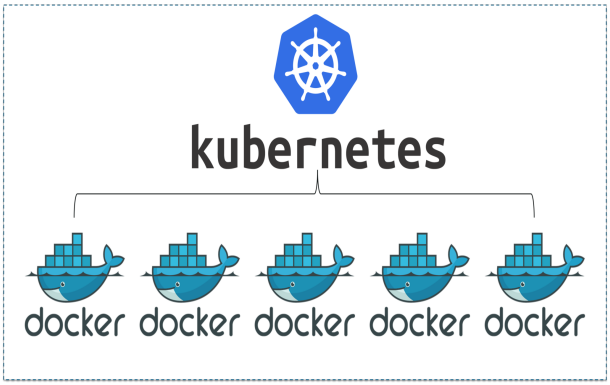


Figure 16: Kubernetes in relation to Docker

[12] Kubernetes is an open-source orchestrator for deploying containerized applications [12]. It allows us to automatically scale our application responding to an increase or decrease in traffic. Kubernetes is self-healing, meaning that it will restart containers that fail. All of this comes in very handy when dealing with a large number of containers.

4.4 Rook

Rook is a Kubernetes storage orchestrator (written as a Kubernetes Operator) that turns distributed storage systems into self-managing, self-scaling, self-healing storage services [1].

[0]Source:https://www.edureka.co/blog/kubernetes-tutorial/
[1]Source:https://rook.io/

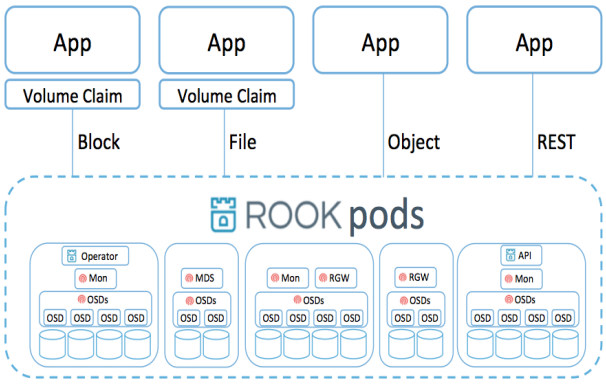


Figure 17: Rook Architecture

It allows use of file, block and block storage and an easy and automated integration for EdgeFs and other storage providers like Ceph and Cassandra just to name a few, and also automates tasks like deployment, provisioning, scaling, monitoring and all the other storage administration related tasks.

4.5 EdgeFs

EdgeFS, like Rook, is also a Kubernetes storage orchestrator (written as a Kubernetes Operator) that ads support for provisioning iSCSI volumes, NFS volumes, or S3 buckets with a high throughput and low latency access. It includes immutable self-verifying location-independent metadata that references self-validated location-independent payloads. It can span an unlimited number of geographically independent sites and at each location deploy an EdgeFs segment as a container. Data is stored using blocks with a strong cryptographic hash to identify, verify and retrieve. To avoid transmission of duplicate payloads EdgeFs hashes the storage block before requesting it. Data is placed dynamically by routing the request to a data group and negotiating within that group to place the new block on the least loaded target. To find the least loaded target it uses an EdgeFS FlexHash which is a table that resides in the server memory of the local site and is automatically discovered. All data stored is immutable and versioned, since changing an object would result in a completely different hash code.

[1]Source:https://blog.wescale.fr/2017/08/28/rook-comment-avoir-du-stockage-distribue/

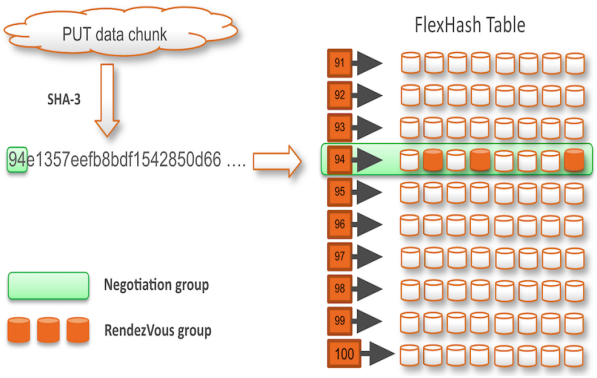


Figure 18: EdgeFs: Representation of data storage

Leveraging local site resources and presenting them as highly available, geographically dispersed cluster segments with immutable data structure design, dynamic data placement, built-in multi-protocol storage gateway and highly scalable, share-free architecture for local sites results in outstanding performance for edge computing.

4.6 CephFs

[14] Ceph is a cutting edge, open source, distributed data storage technology. It is based on self-healing, intelligent object storage devices (OSDs), a combination of CPU, network interface local cache and underlying disk space [14].

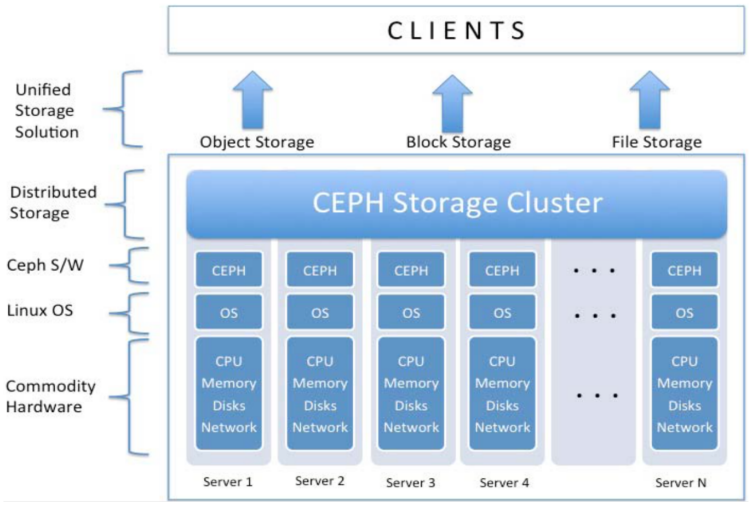


Figure 19: Ceph Cluster Overview

[1]Source:https://medium.com/edgefs/securing-and-deduplicating-the-edge-with-edgefs-bd93e7f786de

[0]Source:https://medium.com/@pk0752/ceph-the-next-generation-store-67f7c51780d3

Objects are the base unit of storage in Ceph. The block storage interface is an abstraction layer that allows users to view a large number of objects as a virtual block device, to which the user can write to and read from. The Ceph clients can be categorized in 3 different types: apps, host or VM and clients. As shown in the image below.

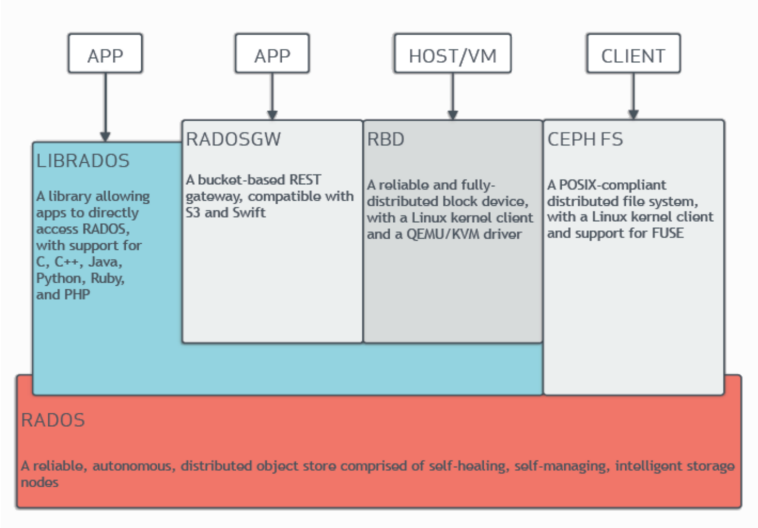


Figure 20: Ceph Architecture Overview

FFor accessing the storage cluster one of four ways can be used, through libRADOS, RADOSGW (HTTP REST gateway for the RADOS object store), RBD (block storage) or CephFs. The libRADOS API enables interaction with Ceph Monitor (maintains master copy of the cluster map) and Ceph OSD Daemon (stores data as objects in a storage node).
CephFs is a file-system built on top of Ceph's distributed object store, RADOS. It is highly scalable, having in mind that it allows clients to directly read from and write to all OSD nodes, and since it is a shared file-system multiple clients can work at the same time. By providing a cluster of Ceph Metadata Servers (manages metadata related to files stored on the Ceph File System), which consists of one active server and multiple standby servers, high availability is achieved. If the active server terminates, on standby servers activate allowing client mounts to continue working through a server failure [2].

[0]Source:https://docs.ceph.com/en/latest/architecture/
[2]Source:https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/2/html/ceph_file_system_guide_technology_preview/what_is_the_ceph_file_system_cephfs

5 Methodology and Implementation

5.1 Setting up GCP and Kubernetes

Using EdgeFs in combination with Rook on Google Cloud Platform we will configure a 4-node Edgefs cluster and use FIO to test overall I/O performance characteristics: file, block, and object.

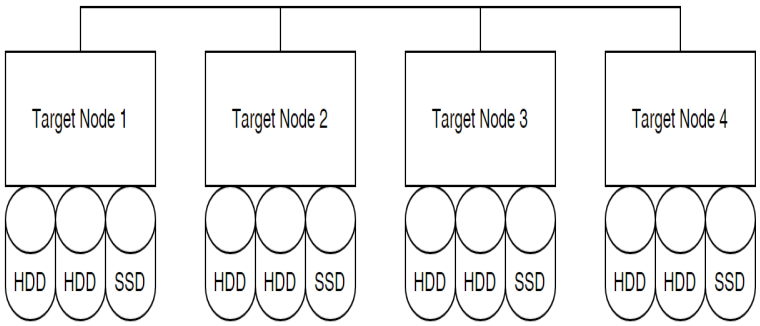


Figure 21: Deployment representation

Target Nodes 1-4 represent data nodes each serving disks. There are 3 different possible combinations:

- All HDDs.
- A hybrid system with HDDs and SSDs, where metadata is offloaded to SSD.
- All SSDs.

As illustrated above, our configuration will be a hybrid one consisting of 2 HDDs, used for capacity, and one SSD for metadata. As illustrated above, our configuration will be a hybrid one consisting of 2 HDDs, used for capacity, and one SSD for metadata. To configure the Kubernetes Cluster in an optimal way, certain rules need to be applied. Unfortunately, at the time of writing this thesis, the EdgeFs git repository and Rook documentation page, which together held 90% of the documentation for EdgeFs, have

been taken down due to a DCMA take-down request issued by DataDirect Networks. Therefore, we will be referring to a Medium article written by Dmitry Yusupov, one of the creators behind EdgeFs, for configuration guidelines [1]. Yusupov states that our target nodes should have 2 CPU cores per SDD (one SSD in our case) and 2GB of memory per device plus an additional 2GB for nodes' other functions (8GB in our case since we have 3 devices). So, we will create a Kubernetes Cluster with 2 node pools, one containing 4 nodes (Target Nodes) and one containing one gateway node. Once configured and deployed our Kubernetes Cluster looks like this.

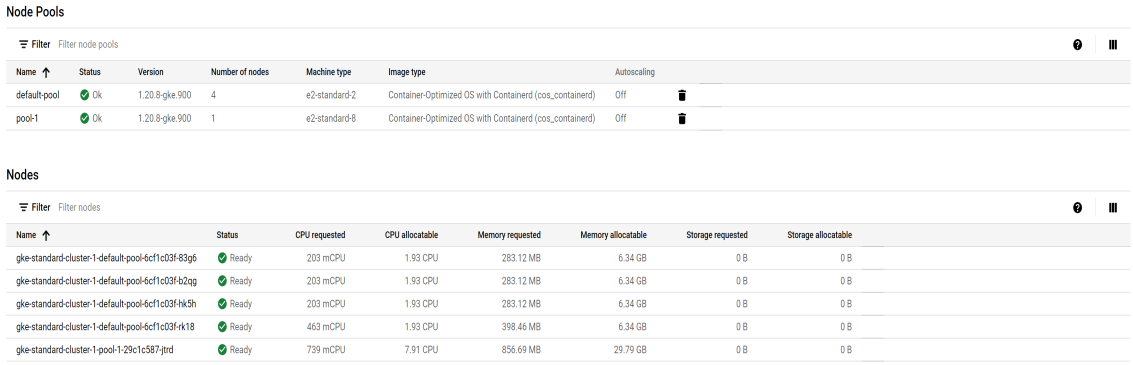


Figure 22: Console View of Kubernetes Cluster

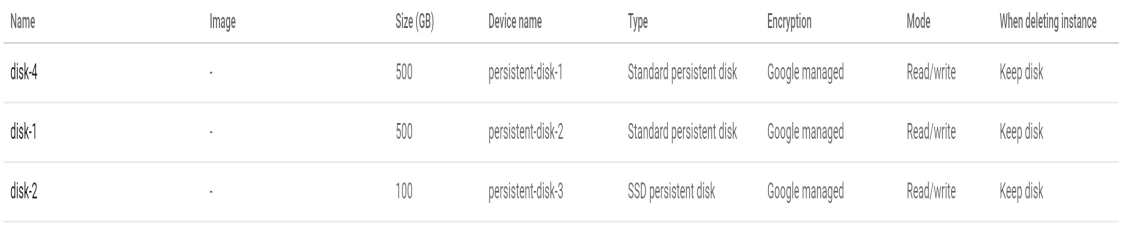


Figure 23: Console View of disk configuration in one node

5.1.1 Deploying EdgeFs

EdgeFs is deployed together with Rook to accomplish that we first need to clone the Rook git repository. Due to the DCMA take-down we will have to use an older non-official version of the Rook repository from Dmitry Yusupov's GitHub page. The deployment of Rook and EdgeFs is handled by YAML files, configuration files most commonly used in applications where data is being stored or transmitted, written in YAML - a human-readable data-serialization language [2]. File operator.yaml will create our EdgeFs operator and automatically discover and allocate our pods, resulting in.

[1] Source:https://medium.com/edgefs/edgefs-cluster-with-rook-in-google-cloud-2dabe954cda6
[2] Source:https://en.wikipedia.org/wiki/YAML

```
milan_nikolic_f@cloudshell:~$ kubectl get po -n rook-edgefs-system
NAME                                  READY   STATUS    RESTARTS   AGE
rook-discover-9rbmp                   1/1     Running   0          9h
rook-discover-h484z                   1/1     Running   0          9h
rook-discover-kppm9                   1/1     Running   0          9h
rook-discover-n7787                   1/1     Running   0          9h
rook-discover-t2bv4                   1/1     Running   0          9h
rook-edgefs-operator-5f7ffc8df4-9xfcj 1/1     Running   0          9h
```

Figure 24: Cloudshell list of pods created by operator.yaml

By using another YAML file, cluster.yaml, we can create our target and gateway pods, resulting in.

```
milan_nikolic_f@cloudshell:~$ kubectl get po -n rook-edgefs
NAME                                READY   STATUS    RESTARTS   AGE
prometheus-operator-b7b5bc8c6-85qc9 1/1     Running   0          4h51m
rook-edgefs-mgr-877b7f674-rhn18     3/3     Running   0          6h50m
rook-edgefs-target-0                3/3     Running   1          6h50m
rook-edgefs-target-1                3/3     Running   0          6h50m
rook-edgefs-target-2                3/3     Running   0          6h50m
rook-edgefs-target-3                3/3     Running   0          6h50m
rook-edgefs-target-4                3/3     Running   0          6h50m
```

Figure 25: Cloudshell list of pods created by cluster.yaml

The interesting thing here is that target and gateway nodes look the same. In fact, the only difference between them is that gateway pods don't serve disks.
The next important construct we need to initialize is FlexHash [13]. FlexHash consists of dynamically discovered configuration and checkpoint of accepted distribution table. FlexHash is responsible for I/O direction and plays an important role in dynamic load balancing logic. It defines so-called Negotiating Groups (typically formed across zoned 8-24 disks) and final table distribution across all the participating components, e.g., data nodes, service gateways, and tools [13]. What this means is that every low level I/O will use FlexHash to negotiate delivery over the network.

```
root@rook-edgefs-target-0:/opt/nedge# efscli system fhtable print
failure_domain 1
numrows 16
vdevcount 20
pid 1245895
leader 0
genid 1628567544457667
from_checkpoint 1
zonecount 0
servercount 5
```

Figure 26: Printout of FlexHash table

5.1.2 Deploying CephFs

For deploying CephFs we will use the same Google Cloud Platform and Kubernetes configuration as we did for EdgeFS. Afterwards, we will test read and write performance using the included rados test command for benchmarking.
For deploying CephFs we will use the example yaml files from the official Rook GitHub page [3].

```
kubectl create -f crds.yaml -f common.yaml
kubectl create -f operator.yaml
```

These two commands will first create the CRDs, common resource, and finally Operator deployment. After our Operators are deployed, we run the two commands displayed below to deploy our Cluster and list all nodes to check if deployment is successful.

```
kubectl create -f cluster.yaml
kubectl -n rook-ceph get pod
```

NAME	READY	STATUS	RESTARTS	AGE
csi-cephfsplugin-6zm7b	3/3	Running	0	26h
csi-cephfsplugin-lpkjb	3/3	Running	0	26h
csi-cephfsplugin-provisioner-7dcc8ff54d-jslj2	6/6	Running	0	26h
csi-cephfsplugin-provisioner-7dcc8ff54d-qfkx4	6/6	Running	8	26h
csi-cephfsplugin-rw6j4	3/3	Running	0	26h
csi-cephfsplugin-tpxd2	3/3	Running	0	26h
csi-rbdplugin-f6rwn	3/3	Running	0	26h
csi-rbdplugin-ls9bk	3/3	Running	0	26h
csi-rbdplugin-nqrhv	3/3	Running	0	26h
csi-rbdplugin-provisioner-6799bd4cb7-9wzc7	6/6	Running	0	26h
csi-rbdplugin-provisioner-6799bd4cb7-lpvmp	6/6	Running	8	26h
csi-rbdplugin-q82wd	3/3	Running	0	26h
rook-ceph-crashcollector-638182bdaf1c2b011254cfb9ce0af28f-7bv97	1/1	Running	0	26h
rook-ceph-crashcollector-8f132e1bf4a64ce14b130dc272c122ae-q7tqb	1/1	Running	0	26h
rook-ceph-crashcollector-ec84db04b2b324a90f7440d101c22a5b-kfzqb	1/1	Running	0	26h
rook-ceph-crashcollector-gke-cluster-1-pool-1-81de930e-w0qmt8g4	1/1	Running	0	26h
rook-ceph-mds-myfs-a-5648c959cc-hxgmt	1/1	Running	0	24h
rook-ceph-mds-myfs-b-64577966dc-8q8t6	1/1	Running	0	24h
rook-ceph-mgr-a-57cb77584b-cv7jb	1/1	Running	0	26h
rook-ceph-mon-a-bd5f7d5c4-kbjwt	1/1	Running	0	26h
rook-ceph-mon-b-867c494bb9-ghbsv	1/1	Running	0	26h
rook-ceph-mon-c-f57b6c94d-n2nlc	1/1	Running	0	26h
rook-ceph-operator-75c6d6bbfc-5j9mt	1/1	Running	0	26h
rook-ceph-osd-0-b9cf7ffd-mqbg2	1/1	Running	0	26h
rook-ceph-osd-1-7dd784968-v2cws	1/1	Running	0	26h
rook-ceph-osd-2-5b56f79c7-vcq2l	1/1	Running	0	26h
rook-ceph-osd-3-5964447c7-xxmrf	1/1	Running	0	26h
rook-ceph-osd-4-7d665b4765-b4l15	1/1	Running	0	26h
rook-ceph-osd-5-598fbc5c95-26mvp	1/1	Running	0	26h
rook-ceph-osd-prepare-gke-cluster-1-default-pool-8353aaab-8cf7z	0/1	Completed	0	37m
rook-ceph-osd-prepare-gke-cluster-1-default-pool-8353aaab-dd9nq	0/1	Completed	0	37m
rook-ceph-osd-prepare-gke-cluster-1-default-pool-8353aaab-sr4vh	0/1	Completed	0	37m
rook-ceph-osd-prepare-gke-cluster-1-pool-1-81de930e-w0qt-j4crq	0/1	Completed	0	37m
rook-ceph-tools-78cdfd976c-l5l56	1/1	Running	0	26h

Figure 27: Cloudshell list of nodes created by cluster.yaml

[3]Source:https://github.com/rook/rook/tree/release-1.7

The most notable and important nodes are the `rook-ceph-mon`, `rook-ceph-mgr`, `rook-ceph-osd` and `rook-ceph-mds`. The Ceph monitors (`rook-ceph-mon`) control all the metadata that is required to store, retrieve and keep data safe. Each monitor has a unique, immutable and static identity which is its IP address. It contains the master copy of the storage cluster map including the storage cluster topology. The Ceph Manager daemon (`rook-ceph-mgr`) provides monitoring and an interface for external monitoring and management systems. Data is stored on the Ceph Object Storage Device which runs the `rook-ceph-osd` daemon and interacts with logical disks attached to the node. The Ceph Metadata Server (`rook-ceph-mds`) manages metadata related to files stored on the Ceph File System (CephFS) and coordinates access to the shared storage cluster [4].

```
kubectl create -f toolbox.yaml
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

The above commands will initialize and connect to our toolbox which contains the `rados test` command for benchmarking.

[4]Source:https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/4/html-single/installation_guide/index

6 Evaluation and Results

Now that everything is configured and deployed, we can test our performance characteristics. For EdgeFS testing we will use its FIO integrated I/O engines. For CephFS we will use its integrated rados test command.

6.1 EdgeFS

FIO is an I/O tester that will allow us to quickly simulate a given workload by spawning a number of thread and processes doing a particular type of I/O action which we will define.

6.1.1 Test Description

We will run two test cases:

- the first one will compare IOPS (Input/output operations per second) between provisioned HDD without EdgeFs and with,
- the second will focus more on read and write speed.

```
[global]
ioengine=ccowvolaio
buffered=0
direct=1
buffer_compress_percentage=50
dedupe_percentage=87
allrandrepeat=0
refill_buffers
norandommap
randrepeat=0
bs=32K
size=10G
io_size=10G
rw=randrw
rwmixread=80
chunk_size=32768
bucket=cltest/test/test-bucket-32k
group_reporting=1
thread=1
[file1]
numjobs=16
```

In our first test we specify how compressible our data is (50%), the de-duplication factor 87% (identifies extra copies of data and deletes them) and will generate 16 files. The second test actually depends on the rwmixread flag. First, we will run the test with

rwmixread set to 0, meaning 100% writes, then set to 100, meaning 100% reads. For both tests we will create 4 different buckets, each with 8 jobs and created objects equally distributed among the target nodes. Compression and de-duplication configuration remain the same as in previous test. Data replication count is 1.

```
[global]                              tenant=test
ioengine=ccowobj                      repcount=1
buffered=0                            group_reporting=1
direct=1                              thread=1
buffer_compress_percentage=50
dedupe_percentage=87                  [\\.\bk1]
allrandrepeat=0                       numjobs=8
refill_buffers
norandommap                           [\\.\bk2]
randrepeat=0                          numjobs=8
bs=2M
size=2G                               [\\.\bk3]
io_size=2G                            numjobs=8
rwmixread=0
rw=randrw                             [\\.\bk4]
cluster=cltest                        numjobs=8
```

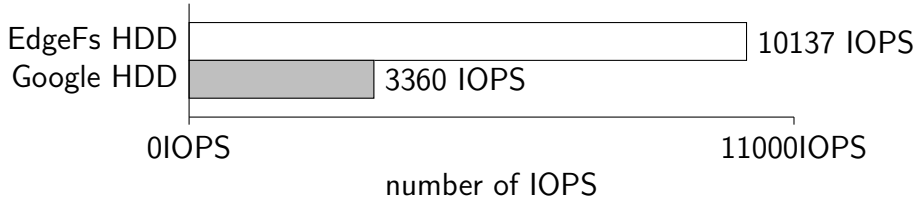
Now that we have defined our tests, let's see, analyze and chart our results.

6.1.2 Results

Here is the generated output of the first test:

read: IOPS=8300, BW=241MiB/s (260MB/s)(128GiB/573205msec)
write: IOPS=1837, BW=63.2MiB/s (70.1MB/s)(32.3GiB/462029msec)

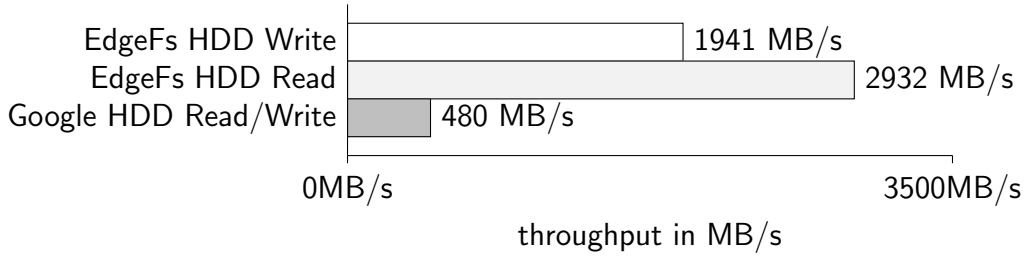
While provisioning an HDD disk in Google Cloud Platform we can see that the disks are rated for 350 IOPS read and 700 IOPS write. With an 80/20 workload, which we have specified in our test file, we can calculate that the maximum IOPS for 8 HDDs is approximately 3360 IOPS (80% read 6.4 HDD's * 350 IOPS + 20% write 1.6 HDD's * 700 IOPS). Let's compare that visually to what we've got.



This 300% improvement in IOPS is mostly due to de-duplication and compression but also offloading metadata to the SSD.
Now let's take a look at the generated output of the second test:

write: IOPS=932, BW=1849MiB/s (1941MB/s)(64.0GiB/34061msec)
read: IOPS=1391, BW=2827MiB/s (2932MB/s)(64.0GiB/22683msec)

Similar to the explanation of the first test, while provisioning HDDs disk in Google Cloud Platform we can see that the disks are rated for 60MB/s read and write. But now we are running the same test twice, once for 100% write than 100% read, that means that our 8 HDD without EdgeFs can read and write at a maximum of 480MB/s (8 HDD's * 60MB/s).



Here we can see an increase of 400% for read and 300% write, this is again due to the smart de-duplication and compression of data chunks before sending it over to the network gateway.

6.2 CephFs

Testing performance on a RADOS storage cluster using the Ceph toolbox and rados test command is simple and fast, by executing one write and two types of read test.

6.2.1 Test Description

As stated above we execute 3 different types of tests.

```
rados bench -p testbench 30 write -t 64 -b 10000000 --no-cleanup
rados bench -p testbench 30 seq
rados bench -p testbench 30 rand
```

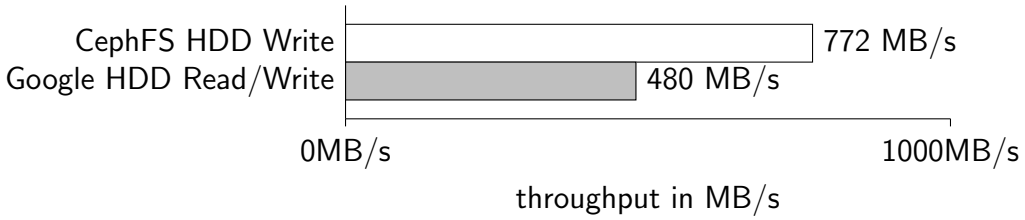
The first command runs a test with 64 concurrent writes of 10MB bytes to objects of size 10, lasting up to 30 seconds. The —no-cleanup flag is important since by default the command will delete objects it has written to the storage pool. Similarly, the next commands run a test sequential and random read, respectively.

6.2.2 Results

Now that we have our test defined, let's analyze the result. Beginning from the first command, a write test.

```
Cur ops  started  finished  avg MB/s  cur MB/s  last lat(s)  avg lat(s)
   4       152       148    145.5037   772.205     4.05607     11.0554
---------------------------------------------------------------------------
Average Latency(s):     10.8479
Stddev Latency(s):      3.56829
Max latency(s):         14.9274
Min latency(s):         0.836277
```

From the above results we can see that the maximum write speed we've got is 772.205 MB/s compared to the theoretical HDD maximum of 480MB/s, calculated in a previous chapter, that is an increase of more than 50%. Minimum latency is very good at ¡1s but we have a high maximum latency of almost 15s. Let's visualize that with a graph.



Next we will look at the sequential read and random read commands.

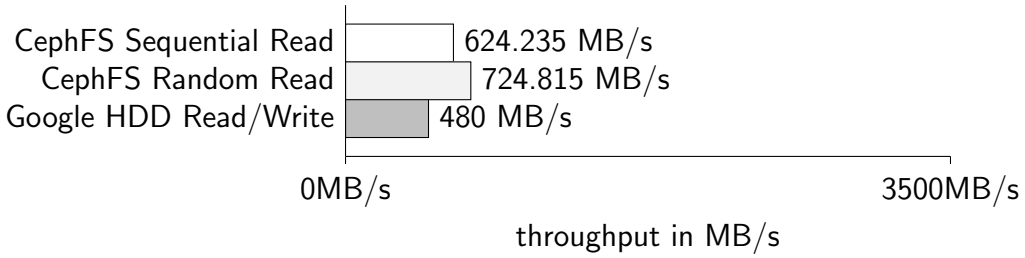
```
Sequential Read:
Cur ops  started  finished  avg MB/s  cur MB/s  last lat(s)  avg lat(s)
   16      117       101     503.81   624.235     0.007545    0.135894
---------------------------------------------------------------------------
Average Latency(s):     0.164992
Max latency(s):         0.547845
Min latency(s):         0.007545

Random Read:
```

```
Cur ops  started  finished  avg MB/s  cur MB/s  last lat(s)  avg lat(s)
   15      400       385     558.276   724.815     0.097046    0.318651
---------------------------------------------------------------------

Average Latency(s):     0.320861
Max latency(s):         1.57361
Min latency(s):         0.0830519
```

Similarly to the first test we can see that the maximum sequential read speed at 624.235 MB/s is around 50% faster than the theoretical maximum of 480MB/s while the random read at 724.815 MB/s is around 60% faster, with very low latency for both sequential and random reads. Let's visualize that with a graph.



7 Conclusion

As demonstrated above, by running EdgeFs in combination with Rook, Kubernetes on Google Cloud Platform we can easily and significantly enhance the performance of HDDs, SSDs available on GCP. Similarly, we can do the same using CephFs in combination with Rook, Kubernetes on Google Cloud Platform. Regarding further work I would like to implement one or both of these technologies on the previously chosen Raspberry Pie device, connecting them and managing them together and comparing their performance in real world scenarios with real data. Unfortunately, this will not be possible with EdgeFs since, while writing this article, it has been abandoned and unlikely to have any future. Without an official supported repository and up to date documentation, implementation of any kind would be very difficult to say the least. Ceph, on the other hand, is a well maintained and documented project and I look froward on implementing it with Raspberry Pie devices.

8 Povzetek naloge v slovenskem jeziku

Zaključna projekta naloga na začetku predstavi zgodovino interneta in njeno evolucjio od WEB 1.0 do WEB 3.0, potem opiše tradicionalno centralizirano shranjevanje, porazdeljeno shranjevanje in razlike med njimi. Nadaljuje z uvodom in opisom naprav IoT, ter z predstavitvijo najbolj popularnih naprav, minimalnih zahtev in končno izbiro naprave IoT glede na te zahteve. Kot uvod za opis implementacije sledi predstavitev tehnologij ki bo v uporabi (Docker, Kubernetes, Rook, EdgeFs in CephFs). Zaključna naloga se konča s primerjavo EdgeFs v CephFs implementacijah z navadnimi diski Google Cloud Storage (brez EdgeFs ali CephFS) in pregledom rezultatov narejenih testov.

9 Bibliography

[1] CURRAN, James; FENTON, Natalie; FREEDMAN, Des; in Misunderstanding the Internet, *Routledge*. 2012. *(Quoted on page 1.)*

[2] BERNERS-LEE, Tim; BRAY, Tim; CONNOLLY, Dan; COTTON, Paul; FIELDING, Roy; JAECKLE, Mario; LILLEY, Chris; MENDELSOHN, Noah; ORCHARD David; WALSH, Norman; WILLIAMS, Stuart, *Architecture of the World Wide Web, Volume One*, `https://www.w3.org/TR/webarch`. (Date of viewing: 06. 08. 2021.) *(Quoted on page 1.)*

[3] WEISONG, Shi; JIE, Cao; QUAN, Zhang; YOUHUIZI, Li; LANYU, Xu in Wielding new media in Web 2.0: exploring the history of engagement with the collaborative construction of media products, New Media Society. *2009*, sprejeto v objavo. *(Quoted on page 2.)*

[4] CHOUDHURY, Nupur, World Wide Web and Its Journey from Web 1.0 to Web 4.0. *International Journal of Computer Science and Information Technologies, 2014*, sprejeto v objavo. *(Quoted on page 2.)*

[5] MCCLELLAND, Calum, *IoT 101 An Introduction to the Internet of Things*, Leverege LLC, 2018. *(Quoted on page 9.)*

[6] COOK, Sam, *60+ IoT statistics and facts*, `https://www.comparitech.com/internet-providers/iot-statistics/`. (Date of viewing: 14. 07. 2021.) *(Quoted on pages 9 in 10.)*

[7] SHIVAM, Arora, *IoT Explained: What It Is, How It Works, Why It Matters*, `https://www.simplilearn.com/what-is-iot-how-and-why-it-matters-article`. (Date of viewing: 14. 07. 2021.) *(Not quoted.)*

[8] MCNELLY, Andrew; GRANT, Martin; CHANG, Henry; COOKE, Larry; HUNT, Merrill; LEE, Todd in Surviving the SOC Revolution: A Guide to Platform-Based Design, *Kluwer Academic Publishers*. 1999. *(Quoted on page 10.)*

[9] WEISONG, Shi; JIE, Cao; QUAN, Zhang; YOUHUIZI, Li; LANYU, Xu in Edge Computing: Vision and Challenges, IEEE Internet of Things Journal. *2016*, sprejeto v objavo. *(Quoted on page 14.)*

[10] S in h, a. *r*, sprejeto v objavo.ad Agarwal Matthai Philipose Victor Bahl AGARWAL, Sharad; PHILIPSE, Matthai BAHL, Victor Vision: The Case for Cellular Small Cells for Cloudlets International Workshop on Mobile Cloud Computing Services 2014 *(Quoted on page 14.)*

[11] VAQUERO, Luis M.; RODERO-MERINO, Luis; CACERES, Juan; LINDNER, Maik in A Break in the Clouds: Towards a Cloud Definition, ACM SIGCOMM Computer Communication Review. *2009*, sprejeto v objavo. *(Quoted on page 15.)*

[12] BURNS, Brendan; BEDA, Joe; HIGHTOWER, Kelsey; in Kubernetes Up Running: Dive into the Future of Infrastructure, *O'Reilly Media*. 2019. *(Quoted on page 17.)*

[13] *Setting up FlexHash and Site root object* , `http://edgefs.io/docs/Quick-Start---Initialization.html`. (Date of viewing: 10. 08. 2021.)

[14] BORGES, Goncalo; CROSBY, Sean; BOLAND, Lucien in CephFS: a new generation storage platform for Australian high energy physics, Journal of Physics: Conference Series. *2017*, sprejeto v objavo. *(Quoted on page 23.)*

(Quoted on page 19.)