

Preventing Insider Attacks in Databases

Type: Thesis **Received:** May 30, 2023 **Published:** June 06, 2023

Citation:

Vijay K Chaurasiya., et al. "Preventing Insider Attacks in Databases". PriMera Scientific Engineering 3.1 (2023): 03-32.

Copyright:

© 2023 Vijay K Chaurasiya., et al. This is an open-access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Sourav Mishra and Vijay K Chaurasiya*

Dept. of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj, India

*Corresponding Author: Vijay K Chaurasiya, Dept. of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj, India.

Abstract

Many web applications that rely on centralized databases face vulnerabilities to insider attacks. While these systems implement multiple layers of security measures against external hackers, they may overlook the threat posed by employees who are already within these security layers and have access to privileged information. Users with administrative privileges in the database system can potentially access, modify, or delete data, while also manipulating corresponding log entries to erase any evidence of tampering, making detection nearly impossible. While one approach could involve developing methods to detect and trace such attacks, along with recovering the original data, this report takes a different perspective. Instead of focusing on detection and recovery, we explore a new direction: ensuring that attacks do not occur in the first place. By establishing a system that comprehensively safeguards data integrity, the need for detection, tracing, and recovery can be minimized or eliminated. This report investigates the prevention of insider attacks on databases by utilizing Bluzelle, a NoSQL database that offers decentralized database solutions for decentralized applications.

Keywords: Tampering; Centralized Database Systems; Insider Attack; Detection; Recovery; Integrity; Bluzelle

Introduction

Bradley Manning and Edward Snowden who were a part of the US military and were considered trusted people, stole critical intelligence documents and released them. The worst came later in 2015 when the Office of Personal Management of the US disclosed that someone had acquired personal information of 22 million federal employees, including employees with the highest level of security clearance. Whoever the thief, they put the security of an entire country at risk. Did they get caught by the security measures in place? No.

Here we are talking about preventing attacks coming from people we trust. This can be achieved by monitoring behavior of users and how they interact with the organization's database, analyze the query logs which are records of questions that people ask to the database and the way they ask them. One may think that we are overreacting, we should trust our employees which is completely correct, but it is better to be prepared than being sorry later.

An insider attack is launched by an authorized personnel who is familiar with the network architecture and is aware of the system policies and procedures. There is usually less security against insider attacks because many organizations focus on protection from external attacks. Insider attacks can affect security elements of the system, steal sensitive data, inject trojan viruses in the system, affect availability of the system by overloading storage or processing capacity which will ultimately lead to system crash. Internal intrusion detection systems (IDS) protect organizations against insider attacks but their deployment is far from easy. Employees must be made aware of certain rules and regulations which ensure that no false alarms are raised.

"In 2008, an incident occurred involving Terry Childs, a network engineer employed by the San Francisco Department of Telecommunications and Information Services. During this incident, Childs modified the network passwords, effectively restricting access to FiberWAN for a duration of 12 days. Subsequently, Childs was convicted of felony network tampering. The city of San Francisco incurred significant costs in the process of regaining control of the affected systems, amounting to \$900,000. Moreover, the insider attack had far-reaching consequences, impacting approximately 60 percent of the city's services.



A lot of web or mobile applications that we use today rely on centralized database systems. Normally database servers are secured behind layers of firewalls and certain access control policies are implemented which protect the database from external attacks but do not fare well against insider threats. This is because these policies provide privileged access to certain users thereby enabling them to modify the database entries and remove the corresponding database and system logs which makes detection impossible.

Client Side Database Protocol Attack

Each database vendor essentially has a proprietary network protocol and they use this for communication and commands. This is generally a highly complex and unfortunately very obscure protocol and they often change. This makes them prone to security vulnerabilities and some of the consequences of this are unauthorized access, manipulation. There are ways to mitigate such attacks : one is protocol validation engines which can address even the unknown vulnerabilities i.e. we let normal client generated messages get through and anything that has hidden qualities or features we drop them and the other one is reactive protocol validation measure which is going to address known vulnerabilities and this will check for specific known attacks.

This attack is a client-side vulnerability. A hex editor is used for looking at some Oracle information.

Cygrue fille	EDITI	ON-	[orac	Sec.1	0.4	1										ALC: NO.
File EGE VA	CH V	nia=		D.												
	X B)	8		*	6 34	18.1	E 12	880	2		_					
00206140	27	25	73	27	20	48	4C	53-5F	54	45	52	52	49	54	45	'%s' NLS_TERRITO
00206150	52	59	3D	20	27	25	73	27-20	48	4C	53	5P	43	55	52	RY= "%s" BLS_CUR
00206160	52	45	48	43	59	3D	20	27-25	73	27	20	48	4C	53	SF	RENCY- '%s' NLS_
00206170	49	53	47	SF	43	55	52	52-45	48	43	59	30	20	27	25	ISO_CURRENCY= '%
00206180	73	27	20	48	40	53	SP	48-55	4D	45	52	49	43	5.F	43	s' NL5_NUMERIC_C
00206190	4.0	41	52	41	43	54	45	52-53	30	20	27	25	73	27	20	HARACTERS= '%#'
002061A0	48	4C	53	5.8	43	41	4C	45-48,	44	41	52	3D	20	27	25	NLS_CALENDAR= '%
00206180	73	27	20	48	4C	53	SF	44-41	54	45	SF	46	49	52	4D	S' NLS_DATE_FORM
002061C0	41	54	3D	20	27	25	73	27-20	48	4C	53	5F	44	41	54	AT= "%s" NLS_DAT
00206100	45	SF	4C	41	48	47	55	41-47	45	3D	20	27	25	73	27	E_LANGUAGE= '%8'
00206180	20	48	4C	53	5 F	53	4.F	52-54	3D	20	27	25	73	27	0.0	NLS_SORT= '%s'.
02061F0	0.0	0-0	0.0	00	00	0.0	00	00-00	00	00	00	00	00	0.0	00	
00206200	6B	70	75	62	61	6C	74	73-32	31	0.0	00	0.0	0.0	0.0	00	kpubalts21
00206210	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	
00206220	41	40	54	45	52	20	53	45-53	53	49	4 1	48	20	53	45	ALTER SESSION SE
00206230	54	20	48	40	53	5F	4C	41-4E	47	55	41	47	45	3D	20	T HES_LANGUAGE=
00206240	27	25	73	27	20	4.0	4C	53-5F	54	45	52	52	49	54	4.F	'As' MLS_TERRITO
00206250	52	59	3D	20	27	25	73	27-20	48	40	53	5F	43	55	52	RY- 'NA' BLS_CUR
00206260	52	45	4 E	43	59	3D	20	27-25	73	27	20	41	40	53	SF	RENCY- 'NS' NLS_
00206270	49	53	4.F	SF	43	55	52	52-45	4.0	43	59	3D	50	27	25	ISO_CURRENCY= 'N
00206280	73	27	20	48	40	53	SF	48-55	4D	45	52	47	43	5.F	43	", MER NUMERIC C
0206290	40	41	52	41	43	54	45	52-53	3D	20	27	25	73	27	20	HARACTERS= 'SR'
00206280	48	4C	53	58	43	41	4C	45-48	44	41	52	3D	20	27	25	NL5_CALENDAR= 'N
0205280	23	27	20	48	4C	53	SF	44-41	54	45	5F	46	4F	52	4D	B' NLS_DATE_FORM
02062C0	41	54	3D	20	27	25	23	27-20	48	4C	53	SF	44	41	54	AT- 'NE' NLS_DAT
0206200	45	5.F	4C	41	48	47	55	41-47	45	3D	20	27	25	73	27	E_LANGUAGE= '%s'
002062E0	20	48	4C	53	5.F	53	4.P	52-54	3D	20	27	25	73	27	20	HLS_SORT- 'As'
002062F0	410	4C	53	SP	43	47	4D	50-3D	20	27	25	73	27	00	00	NL5_COMP+ 'NS'
0206300	68	10	75	62	61	6C	74	73-32	34	00	00	00	00	60	00	kpubalts24
00206310	00	00	0.0	0.0	0.0	0.0	0.0	00-00	00	0.0	0.0	00	00	0.0	0.0	**************
0206320	41	4C	54	45	52	20	53	45-53	53	49	4.F	48	20	53	45	ALTER SESSION SE

This is some DLL information that happens to be in hexadecimal format. The information highlighted on the right this is just a simple query and it says ALTER SESSION SET NLS_LANGUAGE etc etc. This query this is actually the login stream for a sql query from a client. The attacker takes advantage of this. He uses his sql client when he logs in to the database and launches the vulnerability and creates a new user for himself.

CYPER FREE	EDIT	ióni -	[onad	Sector 1	0.4	1										Address of the second
File EGE VM	CH. VI	THE R	1.140	P												
	X RD	8.			2.24	LE I	E 17	BBT	5							
00206140	27	25	73	27	20	41	40	53-5F	54	45	52	52	49	54	45	'%s' NLS TERRITO
00206150		-					-			-		-			-	RY- "SH" NLS CUR
00206160	0	100	la	10	1											RENCY- '%s' NLS
00206170	0	a	-ic	10												150 CURRENCY= '%
00206180	C	lie	at 1	Sid	la	10	cuir	Stre	101	-						s' MLS_NUMERIC_C
00206190	-	ile i	il.	DIG		LU	9"	1 Sue	sai							HARACTERS= '%='
002061A0	48	40	53	58	43	41	4-C	45-4E	44	41	52	30	20	27	25	NLS_CALENDAR= '%
00206180	73	27	20	48	4C	53	SF	44-43	54	45	SF	46	48	52	4D	S' NLS_DATE_FORM
002061C0	41	54	3D	20	27	25	73	27-20	48	4C	53	SF	44	41	54	AT- 'SS' NLS_DAT
00206100	45	SF	4C	41	48	47	55	41-47	45	3D	20	27	25	73	27	E_LANGUAGE= "%s"
00206180	20	48	4C	53	5.F	53	4.P	52-54	30	20	27	25	73	27	00	NLS_SORT= '%s'.
002061F0	0.0	00	0.0	-00	00	0.0	00	00-00	0.0	0.0	-00	00	0.0	00	00	**************
00206200	6.0	70	75	62	61	6C	74	73-32	31	00	00	00	0.0	00	00	kpubalts21
00206210	00	00	0.0	00	0-0	00	0.0	00-00	00	00	00	00	00	00	00	
00206220	41	40	54	45	52	20	53	45-53	53	49	47	48	20	53	45	ALTER SESSION SE
00206230	54	20	48	4C	53	SF	4C	41-4E	47	55	41	47	45	3D	20	T NLS_LANGUAGE-
00206240	27	25	73	27	20	4.6	4C	53-5F	54	45	52	52	49	54	4.F	"NE" BLS_TERRITO
00206250	52	5.9	3D	20	27	25	73	27-20	4.E	4C	53	SF	43	55	52	RY- 'NE' NLS_CUR
00206260	52	45	48	43	59	3D	20	27-25	73	27	20	48	40	53	SF	RENCY- 'NB' NLS_
00206270	49	53	41	2.8	43	55	52	52-45	48	43	59	3D	20	27	25	IBO_CURRENCY
00206280	73	27	20	48	4C	53	SF	48-55	4D	45	52	47	43	SF	43	" NLS_NUMERIC_C
00206290	40	41	52	41	43	54	45	52-53	3D	20	27	25	73	27	20	HARACTERS- 'AR'
00206280	4.8	40	53	2.1	43	41	40	45-48	44	41	52	3D	20	27	25	NL5_CALENDAR= 'N
00206280	23	27	20	48	40	23	38	44-41	24	95	24	46	25	22	40	B HLS_DATE_FORM
002062C0	43	24	30	=0	47	- 22	13	21-20		+C	23	OF		41	5-4	AT- SE MLS DAT
00206200	45	25	+C	41	25	27	22	41-47	45	30	20	27	4.5	12		E LANGUAGE 'SS'
00206280	20	48	4C	22	25	22	25	52-54	30	20	23	25	1.2	122	20	HLS_SORT- 'SE'
002062F0	48	90		28	19.3	100	40	50-ND	2.0	100	2.5	1.00			00	HLM CLMP- 53
00206300	68	70	13	62	01	PC.	14	73-32	24	00	00	90	00	-0	00	spuparease
00206310	-00	00	00	00	40	00	20	00-00	00	00	00	99	00		00	
00206320	41	40	54	45	52	20	2.2	45-53	5.3	49	40	48	50	2.3	45	ALTER SESSION SE

By modifying information on the client the attacker creates a new user on the database. This is very interesting because the actual user login does not have to even be successful. This can be a completely unsuccessful login. It is the process of logging in that will actually take advantage of the vulnerability.

Now this works in the unpatched version of Oracle 10 because the database user in Oracle actually operates with DBA privileges. So if the attacker causes that user to fail via say a buffer overflow for example then he can inject code inside this to actually create a new user or do any number of things that DBA might want to do. The attacker makes some changes here by taking this Oracle 10 file and this client log on stream information and it brings over here.



He then edits this to create a user called hacker and set that user's privileges to DBA.

File Fort Me	~ ~		244		04104				-		-					
	X. Ro 1	8 -	1.19		2 34	UP I	Ed	880	;							
0206180	73	27	20	48	40	53	5F	44-41	54	45	5F	46	4.F	52	4D	S" NLS_DATE_FORM
02061C0	C	100	at	- 1	lee			Ha	ak	or						AT- 'SS' NLS_DAT
02061E0	-	ALC:	au		150			110	Ch	e.						NLS SORT- 'Sa'.
02061F0	S	Set	P	ivi	lec	les	÷	DE	AI							
0206200	1.0	1000	10		100	100										kpubalts21
0206210	00	0.0	00	00	00	00	00	00-00,	00	00	00	0.0	00	00	00	
0206220	6.4	65	63	6C	61	72	65	20-61	20	6.8	75	6D	62	65	72	declare a number
0206230	38	2.0	62	65	67	6.9	6E	20-73	65	6C	65	63	74	20	63) begin select c
0206240	6.F	75	6-E	74	28	2A	29	20-69	6 E	74	6F	20	61	20	66	ount(*) into a f
0206250	72	6.F	6D	20	61	6C	6C	5F-75	73	65	72	23	20	77	68	rom all_users wh
0206260	65	72	65	20	75	23	65	72-6E	61	6D	65	20	30	20	27	ore username -
0206270	22	41	43	40	45	22	27	38-20		00	20	01	20	30	20	HACKER'J IL a -
0206280	30	20	14		65	98	20	65-78	22	23	12	12	22	20	0.9	o then execute 1
0206290	60	60	- 22	22	6.9	01	74	65-20	27	22	12	0.0	23	22	22	mediate create
0206280	20	12	10	20	10	20	2.4	20-63	7.0	20	14	20	22	20	65	at First by backs
0206200	22	22	30	20	65	20	44	63-75	24	65	20	20	60	60	65	rit execute imme
0206200	64	2.5	61	74	65	20	27	67-72	61	68	24	20	6.4	62	61	diate forant dba
02062E0	20	74	6.8	20	60	61	63	68-65	72	27	38	20	65	68	64	to hacker's end
02062F0	20	69	6.6	38	20	65	6E	64-3B	20	20	20	20	20	00	00	ifr end;
0206300	6B	70	75	62	61	6C	74	73-32	34	00	00	0.0	00	00	00	kpubalts24
0206310	00	0.0	00	00	00	00	0.0	00-00	00	00	00	0.0	00	00	00	
0206320	64	65	63	6C	61	72	65	20-61	20	68	75	6D	62	65	72	declare a number
0206330	3B	20	62	65	67	69	6E	20-73	65	6C	65	63	74	20	63	; begin select c
0206340	6.F	75	68	74	28	2A	29	20-69	6E	74	6.F	20	61	20	66	ount(*) into a f
0206350	72	6.F	4D	20	61	6C	6C	5F-75	73	65	72	73	20	77	6.8	rom all_users wh
0206360	65	72	65	20	75	73	65	72-68	61	6D	65	20	30	20	27	ore username = '
0206370	48	41	43	481	45	52	27	38-20	6.9	66	20	61	20	3D	20	HACKER'; if $a =$

On the right side, in the highlighted area you can see a query that the attacker is executing to grant DBA to "hacker". So in short the attacker created a user called hacker and set it's privileges to DBA. He placed this in the sql login stream on client side for Oracle client 10 DLL (which is just a file that happens to be on client side).



Then he goes into his operating system and modifies the clients. He uses a copy command to overwrite his hacked version over the pre-existing version.

Login Using	SOL Plus		-
Successful	or Unsuccessful L	ogins Work	
	Log On		- 1
	User Name	scott	-
	oper mana.		- 11
	Password:		
	Host String:	cto-dbi	
	ОК	Cancel	

Then he logs in as the user Scott. Whether the login attemp is successful or unsuccessful doesn't really matter at this point. Then using oracle sql+ and as scott he runs the command to select username from all users whose usernames essentially has a "H".

Crade SQL*Plus He Edt. Search. Options. Heb	_(D) ×
SQL-Plus: Release 10.2.0.1.0 - Production on Thu Sep 28 19:51:46 Copyright (c) 1982, 2003. Gracle. All rights reserved.	1
Connected to: Graele8i Release 8.1.7.0.0 - Production JServer Release 8.1.7.0.0 - Production	
SOL> select username from all_users where username like " $2H\Omega^*$;	
USERNIHE GSESHTTPSGORIN AMPORASORBSUNAUTHENTICATED	
sar>	

He gets the list of user names and you can see highlighted at the bottom we actually now have the user called "hacker".

Insider Database Privilege Abuse

This example uses a java application this is a standard order form. Many organizations have applications that access the database directly through this means as opposed to a web front- end. Sometimes this is called the fat client or thick client and generally they're installed on the end users desktop.

60			
		Button	
MICKEY	Mouse	Eat or Thick Java Client	123400000001005

They connect directly to the database. Thus no web application is required to serve as a bridge between user and database. Depending on how well coded this application is, there could be several vulnerabilities that are actually specific to the application and the backend database. Here we have a malicious insider that has legitimate access to the Java client due to which the simple fact of them(attacker) using it is not going to send up any bells or alarms. So he downloads an application from the web, in this case it is the fast java decompiler.

Can Fig. Som allengen Den Eleft	2 Microsoft Enforced Englance				altia R
Least: Class & Trand V Infinations	Contraction	-62° then 3h free	· Assert Allers B do	alifabile di Berron Man	tripod.
	Teneros Carlo Obfuscation Neme nargitig 3 distantidos aren C.C.+., Sena, GP and Marci, Neme Marci, Sena Marci, Sen	Productive Vision, prover, Jones, Cander, Processed, decomputation, Stranger, scholars, address, scaling, Arit, J. Processer and and processer and processing sparset	Species. Other and an New composition production than composition anglesening and production anglesening and production and	Date Measuffactor The 2nd procession-Jen's Obtaining Condex Systematic & Editor production mercanics	
Jad - the fast JAva	Decompiler				
exception and entropy of the second		Processing and the first succession of the	d une "Discourance that Judi case	and he used in other software set	
NOTE Jud is free for non-commercial use my prior permanent. If you want to use Ju- Peb 29, 2004 Jud hones page moved to be Dev 14, 2003 Anaged contact moal add Nov 23, 2003 the page moved to be (Mo- po 15, 2004) updated (gibAlance)(1) Jud 29, 2003 updated (gibAlance)(2) Jud 29, 2003 updated (gibAlance)(2) Jud 29, 2003 updated Jud(Jud(2)) Au 15, 2003 and and Jud(2) Start (2)	. Instance the version 1.5.4 is 1 Gar a command-the tably or pp-Dware legither comjust hur over to be hardware. The data set of the table of the data set of and Windows 1.	r an a Breary) in your product(c) d) please contact <u>up</u> for condition	-	

This is going to allow him to decompile the application that resides on his desktop. This decompiles the java application and the source code is revealed. There are certain things here that we might find interesting. First we see the *ODBC connect* and then we see *VEDA_App* and *VEDA_Pass*.



It is actually the database user that this application is coded to operate as within the database. For most clients, their apps will use a single database user to connect to the database. This is very much like pooled user accounts between a web application in a database where you might have 10,000 users on an application but they are really interacting with the database is a single pooled user, Oracle shared accounts, sql shared account something of that nature.

The user authenticates themselves with their own credentials when they leverage their thick client their application connecting to the database but the connection is usually operating as the database user that is coded in here i.e. *VEDA_App, VEDA_Pass.* That application is going to maintain all the user rights and all the user restrictions.

Now the application sits on the desktop, the attacker authenticates against that app but that app is interacting directly with the database. If the attacker operates outside of the application and goes directly to the database then the restrictions the application would have applied are no longer there. He can operate with the privileges of the database user account i.e. *VEDA_App, VEDA_Pass* that gives us access to the application.

Pile Edt Query Tools	Winds	Help 日前一の目		Aup Untiller	5a_DB		a 🖬			- 6
IMPERVA DAN(VEDA_ MPERVA DAN(VEDA_ MOPERVA DAN(VEDA_ MOPERVA DAN More		select * fro	n dbo.Use	ca.						2
Vorthwind Dybt John Upt John Upt John Upt										11
Nortwind U pub: U tempdb U tempdb U use Tables Une Tables D use Categ	1	U Username	Password	FirstName	LastName	Email	PhoneNum	C S Address	CONumber	CCDate
Nortwind Dube Unpube Unpube Unerpube Uner Tables Dube Tables Dube Tables Dube Tables Dube Categ dba Count dba droor	1	U Username 1 mickeym	Password	FirstName	LastName	Emmil mickeym@	PhoneNum 555-1111	C S Address	CCNumber 1234000600001005	CCDate 10/05
Northwind Debe Debe User Tables deb Adain db Categ db Categ db db Adain db db Adain db db Adain db db Adain	1	U Username 1 mickeym 2 donaldd	Password minnie scrooge	FirstName Bickey Donald	LastName Nouse Duck	Email mickeym@ donaldd@	PhoneNum 555-1111 555-2222	C S Address 2000 Hol 400 Lake	CCNumber 1234000600001005 123400002000505	CCDate 10/05 05/05
Nortwind Date Date Date Date Urerspace Urerspace doc.org	1 2 3	U Username 1 mickeym 2 donaldd 3 bugsb	Password minnie scrooge carrots	FirstName Rickey Donald Bugs	LastNese House Duck Bunny	Email mickeym@ donaldd@ bugsb@ev	PhoneNum 555-1111 555-2222 555-3333	C S kddress 2000 Nol 400 Lake 230 Carr	CCNumber 1234000600001005 123400020000505 1234000450001205	CCDate 10/05 05/05 12/05
Nortwind Date Dat	1 2 3 4	U Username 1 mickeym 2 donaldd 3 bugsb 4 tazd	Password minnie scrooge carrots tazdevil	FirstName Bickey Donald Bugs Tazmanian	LastNeme House Duck Bunny Devil	Email mickeym@ donaldd@ bugsb@ev tazd@eva	PhoneNum 555-1111 555-2222 555-3333 555-4444	C Skddress 2000 Nol 400 Lake 230 Carr 13.200 Kosl	CCNumber 123400060001005 1234000045001205 123400045001205 123400000501104	CCDate 10/05 05/05 12/05 11/04

Now this malicious insider opens up a tool called query analyzer. This tool that ships with Microsoft sql so logging in using *VEDA_App* and *VEDA_Pass* the attacker is able to perform a simple query, in this case a SELECT * FROM DBO USERS which gives him very critical information like username, password, first name, last name, email etc. He can even get credit card information. He can execute any

you had used the thick client for its intended purposes.

query and it would run as long as the user that he is leveraging i.e. *VEDA_App, VEDA_Pass* has those privileges. In fact this database user account has the privileges to do pretty much anything within this database if we are able to pull up information this sensitive. So it must be remebered that even thick clients need to be monitored even if there are no front-end web applications and this is because monitoring even the raw sql traffic is important to understand how users are interacting with data. This was a user that had legitimate access but who happened to be malicious as well. They were taking advantage of a poorly coded of the Java fat client as well as the poor

Problem Definition



security restrictions on the database back-end so that they could leak much more sensitive information than they would have if they

Let us consider a database application of a university that allows legitimate users to view and/or update grades of students in various courses offered by the institute. The users are required to use their login credentials (user id and password) to access the database. The type of access provided varies according to the user. Students can only view their grades in various subjects where as faculty members can view along with enter/update the grades of students. However there are certain IT administrators who have privileged access to the database which not only enables them to edit the grades of students but also remove the corresponding database and system logs. If this happens then the students may find their grades changed but they won't be able to prove that such changes were not legitimate. Moreover any faculty member also won't be able to attribute such changes to any specific administrator.



Literature Survey

- The paper [2] discusses how data and trust management degrade the efficiency of Intrusion Detection Systems (IDS) and proposes an intersection of blockchain and IDS to protect data integrity and ensure process transparency.
- The paper [4] presents SafeWS, a system that uses cryptographic and software engineering techniques along with data management principles to prevent and detect website hijacking and unauthorized access.
- The paper [1] uses the immutability and tamper-resistant property of blockchains to detect such attacks. They store some metadata corresponding to each database entry on the blockchain. This metadata has a very small overhead but helps verify the database entries when accessed. Whenever a user wishes to update data in the database, the request is broadcasted across the entire network thereby making it public knowledge. If the transaction is confirmed by all the stakeholders then the database is modified otherwise the request is cancelled and the database remains unchanged.



However, tackling the problem of data recovery in the event of the attacker deleting the entire database along with the log files, requires a protocol where the entire blockchain network is constantly monitored by the nodes in a completely random manner. First one node chosen at random monitors the network and is always on the lookout for any change in the blockchain. Then after say 6 seconds another node chosen at random does the same and after say 1 second another node takes its place and after 9 seconds another and so on. The time interval between the nodes' switching should be totally random thereby making it completely impossible for the attacker to exploit this loophole. But while such a protocol may be theoretically sound, it is by no means practical.

Proposed Methodology

We store the data on bluzelle which is a noSQL database that provides decentralized storage of data to dApps as a secured alternative on blockchain. It consists of two tokens: *BLZ and BNT* i.e. external token used for investment and internal token which powers up the network and gives the owners of masternode a reason to stay operational. BLZ is publicly traded ERC-20 token while BNT are private and only meant for bluzelle network. There are two parties in a bluzelle network: *Producers and Consumers*. The producers are like miners who temporarily provide their computing resources to the network for tokens (BLZ or BNT). This producer economy is called a swarm of nodes or meta swarm.



A swarm is a collection of nodes that store the same set of data. Bluzelle network consists of many swarms that can store different sets of data. The consumers are dApps who require an immutable platform that provides decentralized storage. They access the bluzelle resources and pay in tokens (BLZ and BNT). The two tokens can be exchanged in accordance to a 1:1 ratio. A unique index system monitors the network, imposing penalty on the producers that break rules. Producers who perform well are given a chance to become swarm leaders.

Bluzelle has interconnected servers, each of which replicates the same data, all over the world. The data request of a customer is fulfilled by the server which responds first. Even if a server goes down there are plenty of others to handle the request. Deployment of new servers can be done at any time to replace existing ones or to increase geographic availability.

The singleton metaswarm refers to the entire swarm framework while the virtualized metaswarm is a colony of leaf swarms. Data stored by each leaf swarm is shared with the virtualized metaswarm. The data that network gets from a consumer is split up and stored as shards between each leaf swarm. This ensures data replication and security so that no data loss occurs in the event of a single node failure in the swarm. When a consumer wants to retrieve his data, he provides the private key corresponding to its hash value. The encrypted information on bluzelle ensures that no one can access data without its private key.

Implementation of Proposed Work Installation and Start

To get started we first install the Bluzelle JS library by typing npm install bluzelle in our terminal.

:\Users\Son	u\my-bluzelle-project>npm config set https-proxy http://172.31.2.4:8080
C:\Users\Son npm WARN saw npm WARN end npm WARN my- npm WARN my- npm WARN my- npm WARN my-	u\my-bluzelle-project>npm install bluzelle enrow ENDENT: no such file or directory, open 'C:\Users\Sonu\my-bluzelle-project\package.json' mt ENDENT: no such file or directory, open 'C:\Users\Sonu\my-bluzelle-project\package.json' bluzelle-project No repository field. bluzelle-project No RefoxIME data bluzelle-project No license field.
+ bluzelle@@ added 13 pac	.5.522 Kages from 7 contributors and audited 297 packages in 5.045s

Then we proceed to type wmic csproduct get UUID in the terminal to get our system's UUID as it will be required in the codes further.



Heroku

We can use Heroku with bluzelle add-on which automatically creates environment variables, in our application to enable connection to our testnet and provides a dashboard to manage data that our heroku Application committed to testnet.

First we install the Heroku CLI and log in.



Next we deploy an application on heroku and attach the Bluzelle DB add-on to the application as follows:



Next in our project root we include:

oonst h	1170 0 - r	anira(11)	alla!);	
const o	nuzene - no	equire(bluz	ene);	

To connect to a server we give the host and port of a Bluzelle node, and supply a unique *uuid* for our database.

The following is our Bluzelle JavaScript API:

co	nst bluzelle = require('bluzelle');
blı	izelle.connect('ws://testnet.bluzelle.com:51010',
'20	0626C01-BAE0-E111-B128-960908D33EF5');
blı	zelle.create('myKey', 'myValue').then(() =>
1	bluzelle.read('myKey').then(value =>
	console.log(value); // 'myValue'
}).	<pre>}).catch(e => console.log(e.message)); catch(e => console.log(e.message));</pre>

Then we use the three configuration variables in the code of our previously deployed heroku application to connect to the Bluzelle database.



After that we locally replicate configuration variables.



Then we use our configuration variables in our application by replacing the line we use for connecting to testnet i.e.



With environment variables that were set by the add-on:



For Node.js applications we add an entry into the package.json file as follows:



Cryptographic Keys

We need to provide our own private key to use the database. We install OpenSSL and generate a new key by typing the following command:



We receive the following output:

	BEGIN EC PRIVATE KEY
MH	OCAOEEIOFXRKLsWMldZpdDdaYUTLKlDRduwvt/aNa4xvN1kZgxoA
cGE	SuBBAAK
oUQ	DQgAEYVKSk+CjBEuZy1M9G6d8D6Dbv3J7dYDjtrNumB+enmyJV
M0	TPHPGNxto
QgI	bZ11XAWXnUdtOXJHUOkzSyK7yeQ==
	END EC PRIVATE KEY

The private key is entered into the *private_pem* field of the bluzelle constructor. *bluzelle-js* signs off database operations using this key. The decentralized swarm then verifies our signatures to enforce security and authorization. Bluzelle uses the Elliptic Curve Digital Signature Algorithm (*ECDSA*) on the curve *secp256k1* with an *SHA-256* hash which are all built-in to the system.

Database Permissions

Each database is created by one owner who is the only one who can add or remove writers. Initially the number of writers is zero by default. Any user other than a writer trying to modify the database is denied access.

In ECDSA cryptography, the private key is used to generate the public key. The API function of *bluzelle-js* gives user's the public key. Alternately we can get the public key in this way:



Where the file *private.pem* contains the private key.

Caching Code (Setup, Read and Write)

Whenever a client wants to read data he will first try to read a key-value-pair (KVP) from Bluzelle's cache. If the KVP is not in Bluzelle's cache then it is a miss as a result the client will request the data directly from the database.

Firstly we initialize the bluzelle client in our code:

const bz =	bluzelle({
entry: '	vs://testnet.bluzelle.com:51010',
uuid: '2	0626C01-BAE0-E111-B128-960908D33EF5',
private	pem:
'MHQCAO	EEIFNmJHEiGpgITIRwao/CDki4OS7BYeI7nyz+CM8NW3xToAcGBSuBBAAKoU
QDQgAEn	dHOcS6bE1P9xjS/U+SM2a1GbQpPuH9sWNWtNYxZr0JcF+sCS2zsD+xlCcbrRX
DZtfeDmg	D9tHdWhcZKIy8ejQ=='
});	

We do the following to read a record from the Bluzelle cache and fall back to our existing database:



For writing records we write to the cache and to the database:



Experiments and Results Integrating Bluzelle Cache

Suppose I have an online game whose database is in India. If suddenly the game takes off in England then I will have to setup another server and replicate everything. Then if the game takes off in China then I have to do the same thing all over again. This incurs overhead in terms of time as well as money. But with bluzelle there is no need to switch out, replace or remove. It is added as an enhancement to the existing code. There is no need to redesign the app. In fact bluzelle can be integrated into any app in under five minutes.

Check out the *client.js* file below:

	<pre>const temp = appCommandQueue;</pre>
	<pre>clearCache: async () => sendToServer({cmd: 'clearCache'}),</pre>
	uploadImage: async ({id, image}) => sendToServer({cmd: 'uploadImage', id, image}
	requestData: async ({withCache, id, numOfRequests, startRequestId}) => {
32	times(numOfRequests, async (idx) => {
13	let profile;
5 4	<pre>tet tromtache = ratse;</pre>
50 16	const class = Data paul/
10	const start = bate.now(),
in and a second	// With Bluzelle
9	withCache && (profile = await getBZConnection()(dx)(guickread((d),catch(() => undefined));
	// Without Bluzelle
	<pre>profile (profile = await sendToServer({cmd: 'requestData', id, withCache}));</pre>
10	appCommandQueue.push {
10	cmo: upoatekequest,
10	requestId: startBequestId + idy
0	delay: Date now() - start.
	fromCache
	profile: typeof profile === 'string' 2 JSON parse(profile) - profile

In the highlighted portion we connect to Bluzelle and quickread a key-value pair that is identified by the key *id* and set the return value to *profile*. So what we are doing here is what is called a cache aside pattern. I am first asking the cache "hey do you have this value?". If the cache has that value it'll quickly return that value, if it doesn't have that value which is called a cache miss it won't return that value. So hopefully we get a cache hit and *profile* has the object.

Now if *profile* does not have it then it is a cache miss. We next go to this line:

client.js

	<pre>const temp = appCommandQueue;</pre>
27	
28	he second a second of a second for the second secon
29	clearCache: aSync () => sendioServer({cmd; clearCache }), upload[mane: stync ()d (image)) => cendioServer/(cmd; upload[mane) (d (mane))
31	requestData sync (10, image) - senticerter(uma, uproduinage, is, image)
32	times(num)frequests, async (idx) \Rightarrow {
	let profile;
	<pre>let fromCache = false;</pre>
36	const start = Date.now();
37	// With Plutelle
30	<pre>// mich ouzerree // mich ouzerree /</pre>
40	articles of (project - unart generation control (a) (dreated (a)) (dreated ())
	// Without Bluzelle
44	<pre>profile (profile = await sendToServer({cmd: 'requestData', id, withGache}));</pre>
40 46	appCompadDueue puch/4
47	cond: undateRequest.
	data: {
	<pre>requestId: startRequestId + idx,</pre>
	delay: Date.now() - start,
51	fromCache.
	<pre>profile: typeof profile === 'string' ? JSON.parse(profile) : profile</pre>

The highlighted line represents what our client will typically already have or what we might have already programmed into it where we are basically talking to our existing back-end which is probably an existing database server which might be SQL or some sort of middleware but the point is this is what we already have and all we are doing is adding the single line of code(*line 39*) above it saying ask the bluzelle cache first for that value and see if we can quickly get that data back from those out at the edge before we even bother talking to our middleware.

So the benefit here is pretty obvious. We can get a really quick response to getting our data instead of waiting to talk to our server on the back-end. Instead of burdening our server with all the additional overhead of talking to it every time we need data, we just get it from the cache.

Let us keep in mind that sendToServer (*line 44* of *client.js*) is actually calling ultimately something called request data which is on the server side i.e. *server.js*. Using an existing server side cache called Redis which is wrapped around our existing database we are reading our data here from an SQL server or a Mongo server:

server.js

30	await new Promise(resolve => {
	connection = connection [] mysql.createConnection(sqlParams);
33	error ? resolve({error: error.message}) : resolve(results[0]);
34 35	redisset(16, ');
36	
	requestData: async ({withCache, id}) => {
	let profile;
40	Section with redicter (d)
42	(projite - anat redisectio),
	profile = await readFromSQLServer(id);
45	<pre>profile = JSON.stringify(profile);</pre>
40	realsset(id, profile);
48	
	<pre>withCache && bluzelleSet(id, profile);</pre>
51	
52 53 1-	
54	
	nst sqlParams = (
57	user: 'bluzelle'.

The highlighted portion of code above represents our existing middleware. Now we ultimately do not want to hit this very often because readFromSQLServer (*line 44*) is slow, redisGet (*line 41*) is slow and there is also latency of talking to the server. But ultimately whenever we do hit this function we want to add the line highlighted below:

<pre>connection = connection [] mysql.createConnection(sqlParams); connection.query('UPDATE profile set image="s{image" where id="s{id}"*, error ? resolve((error: error.message)) : resolve(results[0]); redisSet(id, '');</pre>
), requestData: async ({withCache, id}) ⇒ { let profile;
<pre>profile = await redisGet(id);</pre>
<pre>if (!profile) { profile = await readFromSQLServer[id); profile = JSON.stringify(profile); redisSet(id, profile); }</pre>
<pre>withCache && bluzelleSet(id, profile);</pre>

What this line does is it basically tells bluzelle "here is this key-value pair, just store it". Within seconds of calling this function bluzelleset(id, profile) the entire bluzelle network is going to have this id and profile value.

It is stored in the cache and that means within seconds any client that asks for that id value here:

client.js

	const temp = appCommandQueue;
	<pre>clearCache: async () => sendToServer({cmd: 'clearCache'}),</pre>
	uploadimage: async (id, image) => sendioserver((cmd: uploadimage; id, image)
	requestuata: async (withcache, id., numurkequests, startkequestid)) => {
	times (unnorkequests, async (idx) => (
	let profile;
2	
6	const_start = Date_now();
	// With Bluzelle ,
	<pre>withCache && (profile = await getBZConnection(idx).guickread(dd).catch(() => undefined));</pre>
	// Without Bluzelle
	<pre>profile (profile = await sendToServer({cmd: 'requestData', ld, withCache}));</pre>
9	appcommanducere.push
	data (
å	requestId: startPennestId + idv
Ā	delay Date now() - start
	fronteche
	profile: typeof profile man istring 2 1500 perceptile) - profile

Is going to get that profile object:

	<pre>const temp = appCommandQueue;</pre>
	<pre>clearCache: async () => sendToServer({cmd: 'clearCache'}),</pre>
	uptoadimage: async ({ld, lmage}) => sendioserver(cmd: uptoadimage, id, image)
	time(numOfRequette, async (idv) => (
	lat profile
	let from ache = false:
	<pre>const start = Date.now();</pre>
	// With Bluzelle
	<pre>withCache && (profile = await getBZConnection(idx).guickread(id).catch(() => undefined));</pre>
	// Without Bluzelle
1	<pre>browned if (profile = await sendToServer((cmd: 'requestData', id, withCache)));</pre>
	appCommandQueue.push({
	cmd: updateRequest,
	data: {
	requestId: startRequestId + idx,
	delay: Date.now() - start,
	Profile tupos profile and istained 2 1500 parce(profile) - profile
2	profite typeof profite === string r Jsuk.parse(profite) : profite

Really quickly. The cache will have been preloaded with that value and now the cache is active for that object.

So for adding data to the bluzelle cache we add this one line to our *server.js* code:

18			
19	δıδı -	bluzelleSet(id, profile);	

And in the *client.js* we add this one line to read:

8	// With Bluzelle	
	<pre>withCache && (profile = await getBZConnection(idx).guickread(id).catch(() => undefined));</pre>	

So we are not changing any existing code, not removing references to our existing database, not changing references to reading from the existing database. Everything is the same just one extra line of code for bluzelle.

Creating Node.js Application with Bluzelle Database

Firstly we install bluzelle library in our system by typing in *npm install bluzelle*. Then we type in the following code in our editor:

cons	<pre>t bluzelle = require('bluzelle');</pre>
	elle.connect('ws://testnet.bluzelle.com:51010',
s jav	ascript_demo');
bluz	elle.create('myKey', 'myValue').then(() =>
2 {	
))	<pre>bluzelle.read('myKey').then(value =></pre>
3	<pre>console.log(value); // 'myValue'</pre>
	<pre>bluzelle.update('myKey', 'newValue').then(() =></pre>
	{
	<pre>bluzelle.read('myKey').then(value =></pre>
	<pre>console.log(value); // 'myValue'</pre>
	<pre>}).catch(e => console.log(e.message));</pre>
	<pre>}).catch(e => console.log(e.message));</pre>
	<pre>}).catch(e => console.log(e.message));</pre>
3) 0	atch(e => console log(e message)):

In line 1 we use the require command to import the bluzelle library. In line 2 we connect to testnet.bluzelle.com which is a bluzelle lovelace testnet and provide a namespace variable javascript_demo. We then create a key-value pair with the value of the key myKey and value myValue. We then read back "my key" we should get back "my value" and we will print that out. We then update the key-value pair with key myValue to newValue. We then read it again and display it on the screen. Now, if everything works as planned we should first see the value myValue printed on the screen followed by newValue. We save the code as *demo.js*.

We go to a CRUD client and validate the namespace javascript_demo and see that those key value pairs are not already there.

Edit View Winde	Biuzelle Database w Help	Editor	• • •
	🔅 bluz	elle	
	Address: testnet.bluzelle.com	Port: 51010	
1	UUID:		
	javascript_demo		
	0		



So we find an empty database.

Then we run the code by typing *node demo.js in our terminal*:



So we see that we got the desired output myValue followed by newValue. Then we go to the CRUD client to verify what appears.

	🔅 bluzelle	
< > III		
+ &		



So we have myKey as expected and then we click on it to check its value.

dit View Window Help	· bluzelle	
< > III		
mytimy	- >	
+ 0		
	Bluzelle Database Editor	
dit View Window Help	Bluzelle Database Editor	© 0 0
dit View Window Help	Bluzelle Database Editor	
dit View Window Help	Bluzelle Database Editor	
dit View Window Help	etrzelle Database Editor etrzelle Database Editor ferevialue I	
dit View Window Help	Bluzelle Database Editor Serviciae I	0 0 0
dit View Window Help < > E A myKey + X G R	Bluzelle Database Editor	• • •

And as expected it has the last value was set to which was newValue.

CRUD Operations using Bluzelle APIs

Let us take a look at an application which will spool up a four node swarm on my machine. It will be a local swarm and the application called swarm spooler. We can see that it has created a four nodes swarm and these four nodes are running a locally on our machine.

G 1	
	Running node with 10: HYVKANKKGII(GAOYFKEEEAAOOgAEb.)xXXG4+KGTSgTL6T8Y7ce8tLzClMiYrkg8+bYC68KupXa18PH4pLgcTT8K2VXxm6Y47Ad5HHgExxcN Etherum Adress 10: exdde2092c73b3b5b1b7ab3b52eac3e66449121a Local 1P Address 127:0.0.1 0 port: 09251 proc
^ e	Used Storage: @ Bytes
	H
	12019-95-91 13:56180.00000 01(1 [0x00000013193560] [otcog] [0013(tp)160] - starting primty alter liner [2019-95-93 13:56180.00000 01(1 [0x00000013193560] [0min.cgp:25]) - starting 12 worker threads [2019-95-94 13:56180.00000 01(1 [0x00000013193560] [into] (min.cgp:22]) - starting 12 worker threads
	Running node vith ID: MFV#LDHNRGZI: (BCAPTFREELABOgARbgZLnTtccuTFR04BC6GG1GGUAO)hYP/oBj42501q6eD5X5q++80/LB1ZET81PJqUD%pH1fD04kwjt Etherwu Adress: ID: subd20932/C3b3a5b1b7ac3b362ac1e8648121a LDcal IP Address: ID: subd20932/C3b3a5b1b7ac3b362ac1e8648121a Token Dalocc: 8929.6 ETM
	Used Storage: @ Bytes
	[2019-65-38 18:56:80,809035 UTC] [0x8000000113f935c8] info] (msin.cop.223) - starting 12 worker threads [2019-65-38 18:56:80,90937 UTC] [0x8000000113f935c8] info] (debug] (ethercum.cop.35) - received response: ["*tatus";1","sessage";"OC", "resui [2019-65-38 18:56:80,9037 UTC] [0x8000000117bc5c8] Info] (bosttrap.perc.rcp;35) - Recaing perst from ./perr.json [2019-65-38 18:56:80,9037 UTC] [0x8000000117bc5c8] Info] [2019-65-38 18:56:80,9037 UTC] [0x8000000117bc5c8] Info] [2019-65-38 18:56:80,90326 UTC] [0x8000000117bc5c8] Info] (region_cop.28) - No monitor is configured; stat will not be collected [2019-65-38 18:56:80,90326 UTC] [0x800000117bc5c8] Info] (region_cop.28) - Using QenstSi I.1.1 11 Sep 2018 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x8000000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x8000000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x8000000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x8000000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x800000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356 UTC] [0x80000117bc5c8] Info] (database_pdftrav/recvec.pp:207) - next_request_sequence: 1 [2019-65-38 18:56:80,20356
	Running mode with ID: MFYvEAYHKoZIzj@CAQYFK4EEAAoDQgAEb21xjV3dr09cbYyHum7vb00BVdfKL5kmuaTspMfYkvl2N3oVUAKUSJWaxaUEpj1/X0Hx2Mbhd6v6JQ0ad Ethereum Address ID: @xdd0d203932763ba5b1b7ae3b362eac3e8d40121a Local Pc.ddcess: JD: @xdd0xdc37.8.0.1
	LOCAL AP ADDRESS: 127.0.0.1
200 200 200 200 200 200 200 200 200 200	19-05-0 13:56:08.07215 UTC [0x00000011235c6] [1010] (0x101001cp;02) - 102 Nonlistor 12 Contagono, 1465 #11101 De Contector 19-05-30 13:56:08.072150 UTC [0x00000011235c6] [1010] (crypto.cp;33) - Using OpenS51 1.1.1 11 Sep 2018 19-05-30 13:56:08.072150 UTC [0x000000110525c6] [1010] (0x10.cp;181) - Using Jan-menory testing storage 19-05-30 13:56:08.072150 UTC [0x000000110525c6] [1010] (0x10.cp;181) - Using Jan-menory testing storage 19-05-30 13:56:08.072150 UTC [0x000000110525c6] [1010] (0x10.cp;181) - Using Jan-menory testing storage 19-05-30 13:56:08.072150 UTC [0x000000110525c6] [1010] (0x10.cp;181) - Using Jan-menory testing storage 19-05-30 13:56:08.072150 UTC [0x000000110525c6] [1010] (0x10.cp;181) - Using Jan-menory testing storage 19-05-30 13:56:08.072160 UTC [0x000000110525c6] [1010] (0x10.asg.poft_service.cp;270) - next_request_sequence: 1 19-05-30 13:56:08.072150 UTC [0x000000111535c8] [1010] (0x10.asg.poft_service.cp;270) - next_request_sequence: 1 19-05-30 13:56:08.072150 UTC [0x000000111535c8] [1010] (0x10.asg.poft_service.cp;270) - next_request_sequence: 1 19-05-30 13:56:08.072150 UTC [0x000000111535c8] [1010] (0x10.asg.poft_service.cp;270) - next_request_sequence: 1 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer 19-05-30 13:56:08.000050 UTC [0x000000110535c8] [1010] (0x010.cp;160) - starting primary altwe timer
	Diam TO:
R	saara ui lunning node viih 1D: MFYvEAYHKoZIz]0CAQYFK4EEAAaOQgAE1LaAP0BwPR1CoKTYxbbZTN58F8b3anPm834sulhibZcIB4hw/oG0YUTry5V9LGBTjlBwozHMr94Yq Ethertum Address 1D: BuddbdZb932c753ba5b1b7a83b362eac3e8d40121a Local IP Address: 1D7.8.0.1
P	umning node wiih 10: MYVEATHEGIZIEGEAAAOQAKILAMPBEAMPICOKTYXEbZTHGBFBbJanPH834sulhibZeIB4hw/oGBYUTrySY9LGBTjlBkozBMr94Yq Eberuma Adress ID: BoddoEd2272758a5b1b7ae3b352eac386448121a Local IP Address IZ7.0.0.1 0 nport 48133 Täken Balance: 48931.0 ETH Used Storage: 0 Bytes
R [20 [20	umning node with 10: MYNGANKGDI;GCANFKGEAAABOQAEILAWPBA-PRICKFYxbbZh68F8b3anPkB34sulhibZeIB4hw/oGPNUTry5V9LGBT][BHoizHMr94Yq Etherum Adfress 10: BoddhC2027x63ba5b1b7a=B353C2eac3eBd40121a On port: 40153 Token Balance: 40891.6 ETH Used Storage: 0 Bytes 119-85-30 18:56:00.000409UTC] [0x0000001102135c50] [info] (mgin.cpp;225] - starting 12 worker threads 19-85-30 18:56:00.00040001002132c5(0) [info] (mgin.cpp;225] - starting 12 worker threads
R [20 R	<pre>umning node wiii 10: Wry4ArMexDII:[0CAPTK4EEAAABOQAKILAWPBM-PMICAKTYxbbZTh6BF8bJanPH834sulhibZeIB4hw/oGPNUTry5V9LGBT][BH0:2HMr94Yq Eberrum Adfress ID: BoddbC2023Cr5Dab5b1b7a=D30362eax3eBd48121a Local IP Address: 127.e.0.1 0 nport: 49153 Taken Balance: 48951.6 ETH Used Strange: 0 Bytes 119-85-39 13:55:08.880449 UTC[[0c8000008120b235c8] [info] (main.cpu:225) - starting 12 worker threads 119-85-39 13:55:08.880449 UTC[[0c8000008120b235c8] [info] (main.cpu:225) - starting 12 worker threads 119-85-39 13:55:08.880449 UTC[[0c8000008120b235c8] [info] (main.cpu:225) - starting 12 worker threads 119-85-39 13:55:08.880449 UTC[[0c8000008120b235c8] [info] (main.cpu:225) - Swarm ID: Umning node with ID: MFV4CMM6ZI2[0C007K4EEAA0000AE8.JV6X64+0073q/L6T8470e8tLzC3MJYmgB+bVC60AupXa18PU4pLgc7T8N2VXxm6Y47Ad5H4gExx Eherrow Address ID: 0bddc2022r2F5ia05b1b7a3b362eac3e8440121a Local IP Address: 127.e.0.1 0 nport: 49151 Taken Balance: 4891.6 ETH Used Storage: 0 Bytes</pre>
R [20 [20 [20 [20 [20 [20	<pre>uuming nod="will" 10: Wry42NH82II;0EQADFA4EEAAe00pdk1LuAPBBAPHICACTYxbb2Th6BF8bJanPH834sulhibZe184hw/oGPNUTry5V9LGBT[184oz8Hr94Yq Enerum Adress 127.e.0.1 0 nport 49153 Taken Balance: 48951.6 ETH Used Storage: 0 Bytes 119-85-30 13:56:80.000460 UTC[0x00000011281c5c0] [info] (main.cpp:225) - starting 12 worker threads </pre>
R [20 [20 [20 [20 [20]] [20]] [20]] [20]]	umning node "ulin 10: MPV4ATMERZI; (GUTVAFK4EAAAODQAK1LAMPBM-MTLGCYTAbZTMGBFBDJanPAB34suthibZeIB4hw/oGMTUTrySV9LGBTjlBMo;2MFV4YQ Eberum Address ID: ModdDd20127C5B305b1b7aE3D352cac286448121a Local IP Address: 127.0.0.1 Mission Balance: 4001.0 ETH Uses Stores: 000 400 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting 12 worker threads 19-45-30 18:568.00040 UTC [Bx000000011021c5c0] [info] (mgin.cpp;223) - starting primary alive timer 19-45-30 18:568.000400 UTC [Bx00000001101595c0] [debug] (mgit.cpp:46) - starting primary alive timer 19-45-30 18:568.000400 UTC [Bx00000001131955c0] [info] (mgin.cpp;223) - starting primary alive timer 19-45-30 18:5688.000040 UTC [Bx00000001131955c0] [debug] (mgit.cpp:46) - starting primary alive timer 19-45-30 18:5688.00000 UTC [Bx00000001131955c0] [info] (mgin.cpp;223) - starting primary alive timer 19-45-30 18:5688.00000 UTC [Bx00000001131955c0] [debug] (mgit.cpp:46) - starting primary alive timer 19-45-40 18:5688.00000 UTC [Bx00000001131955c0] [info] (mgin.cpp;223) - starting primary alive timer 19-45-40 18:5688.000050 UTC [Bx00000001131955c0] [info] (mgin.cpp;223) - starting primary alive timer 19-45-40 18:5688.000050 UTC [Bx00000001131955c0] [info] (mgin.cpp;220) - starting primary alive timer 19-45-40 18:5688.000050 UTC [Bx0000000112135550] [info] (mgin.cpp;223) - starting primary alive timer 14-45-40 18:5688.000050 UTC [Bx00000001213195550] [info] (mgin.cpp;220) - starting primary alive timer 14-45-40 18:5688.000000 UTC [Bx000000012135550] [info] (mgin.cpp;220) - starting p
R [20] [20] [20] [20] [20] [20] [20] [20]	uuning noa ¹ 10: 10: 17945/1862/1;1620/97464E5AA600p4(11_WPBerMP1CGY7kb2Th6BF8bJanPH834sulhib2E1B4hw/oGP1UTry5V91C8T][BH0:2HY94Y9 Ebrerum Address: 127.0.0.1 On port 40153 Taken Balance: 40931.6 ETH Used Storage: 0 Bytes 19-05-30 13:56:00.000460 UTC] [0x000000011281c5c0] [info] (0x1n.cgp:225) - starting 12 worker threads 19-05-30 13:56:00.000460 UTC] [0x000000011281c5c0] [info] (0x1n.cgp:225) - starting primary alive timer 19-05-30 13:56:00.000400 UTC] [0x00000011373556] [debu] Used Storage: 0 Bytes 19-05-30 13:56:00.000400 UTC] [0x00000011373556] [debu] Used Storage: 0 Bytes 19-05-30 13:56:00.00040 UTC] [0x00000011373556] [debu] Used Storage: 0 Bytes 19-05-30 13:56:00.00040 UTC] [0x00000011373556] [debu] Used Storage: 0 Bytes 19-05-30 13:56:00.00040 UTC] [0x00000011373556] [debu] 19-05-30 13:56:00.00050 UTC] [0x000000011373556] [debu] 19-05-30 13:56:00.00050 UTC] [0x00000011373556] [debu] 19-05-30 13:56:00.00050 UTC] [0x0000001147
R [200 [200 [200 [200 [200] [2015] Run [2015] Run	<pre>umning node with 10: WYV4ATMEDIZI:[GUAYTMEETAAADOgAAL1LAMPBM-MPIGCATYNb2ThGBFBDJanPMB34suthibZeIB4hw/oGPUTrySV9LGBTjlBMo2PMITGYV94CGB44B121a Local IP Address: 127.8.6.1 Used Storage: 8 bytes 19-45-38 18:56:88.488459 UTC[[0x48848980112512:56] [inf] (main.cpp:223) - starting 12 worker threads 19-45-38 18:56:88.488459 UTC] [0x48848980112512:56] [inf] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x48848980112512:56] [inf] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x48848980112512:56] [inf] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4884898011371555] [debug] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x488489811371555] [debug] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4884889811371555] [debug] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x488888811371555] [debug] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x488888811371555] [debug] (main.cpp:223) - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x488888811371555] [debug] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x488888811371555] [debug] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888888901157] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888888902556] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888889805556] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888888955556] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888898911578558] [min] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888889805556] [min] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.488459 UTC] [0x4888889805556] [min] [main.cpp:223] - starting primary alive timer 19-45-38 18:56:88.48</pre>
R [20] [20] [20] [20] [20] [20] [20] [20]	<pre>umning nod="will D: prv6ArMegZI;06QV7K4EZAA00004112MP0BerPICGY7bb2Th6BFBDJanPeBJ4suthibZeIB4hw/oGPUTry5V9LGBT [18e22Mr94Yq Editrerum Address DD: edubdD2012/2751b3b1b7acB3b362eac3e6448121a Local IP Address: 127.e.0.1</pre>
R [20] [20] [20] [20] [20] [20] [20] [20]	<pre>umning node with D: PYP4ATMEGIZI:@COVTRAEEAAc00pdEllayPemPICOCTYsb2ThGBFBDJanPeBJ4suthibZeIB4hw/oGPUTrySV9LGBTj1BeozMF149Yq Ebrerum Address D: exbdDsD2127ChBabBlb7acBb352caCeBd48121a Local IP Address: 127.8.6.1 DisedEiner: 48931.6 ETH UsedEiner: 4931.6 ETH UsedEiner:</pre>
R [200 [200 [2015] [20]	<pre>umning near and a set to prove ArthereoIII (BADAPTAGETAAACOOGALLIAMPEMPICOKYNEDITAGETEDJanPeBJ4suthibZeIB4hu/oGPUTrySY9LGT][BeozPMr94Yq Editrerum Adress 127.6.0.1</pre>

<pre>[2019-05-30 18:56:08.080660 UTC] (0x0000001131935 [2019-05-30 18:56:08.080650 UTC] (0x000000011281e5 [2019-05-30 18:56:08.080815 UTC] (0x00000001131935 [2019-05-30 I8:56:08.080815 UTC] (0x00000001131935 [2019-05-30 I8:56:08.080815 UTC] [0x0000001131935</pre>	cð] (debug) (sudit.cpp:46) - starting ⁵ primary alive timer cél línfo (main.cpp:25) - starting 12 worker threads cél línfo (main.cpp:261) -
Running node with 10: MFYvEXYK6Z12/9CA0YFK4EEA Ethereum Address 10: 0xddwd2b932c763ba5b1b7a3b Local IP Address: 127.0.0.1 On port: 49152 Token Balance: 40891.6 ETH Used Storage: 0 Bytes	o00pAEMg2LnYTgcudTFBUV6RGT6IGUvQ0jhYP/o8j42501q6eU5X3q++8U/L812ET8jPJquUXxpM1fD04kux; 362eoc3e8448121a
2019-05-00 18.56:08.000035 UTC 0+00000001114035 2019-05-00 18:56:08.109437 UTC 0+0000000117eb5 2019-05-00 18:56:08.109473 UTC 0+0000000117eb5 2019-05-30 18:56:08.10946 UTC 0+0000000117eb5 2019-05-30 18:56:08.10946 UTC 0+0000000117eb5 2019-05-30 18:56:08.10946 UTC 0+0000000117eb5 2019-05-30 18:56:08.20550 UTC 0+0000000117eb5 2019-05-30 18:56:08.20550 UTC 0+0000000117eb5 2019-05-30 18:56:08.20550 UTC 0+0000000117eb5 2019-05-30 18:56:08.20550 UTC 0+0000000117eb5 Saurn 1D0 Saurn 1D0 Fundaming 10: Fred/MV6221 (0+0000000117eb5 Linerus A: D1: 0+0015732/C730-05017a3B Linerus A: D1: 0+0015732/C730-05017a3B	c8] [info] [muin.ccp;225] - starting 12 worker threads c8] [debug] (etherema.cp:35) - received response; ("status":"1","message":"0K","resi [info] [bootstrap.peers.cp:39] - Reading peers from ./peeri.json c8] [info] [bootstrap.peers.cp:39] - Reading peers from ./peri.json c8] [info] [contifor.cp:65] - No enoitor is configured; stats will not be collected c9] [info] [contifor.cp:65] - No enoitor is configured; stats will not be collected c9] [info] [contifor.cp:65] - No enoitor is configured; stats will not be collected c9] [info] [contifor.cp:65] - No enoitor is configured; stats will not be collected c9] [info] [contifor.cp:65] - next.reposet; sequence; 1 c8] [debug] [dudit.cp:64] - starting primary alive timer c8] [info] [contifor.cp:64] - starting primary alive timer c000pfEb21a; dr0%cbYthm7w000VdfRL5knuaTspHYtkv120Eb400RL5JMaxaUEpj1/X01w2Mbhd6v6300. 3520w25e040E12a

Now we go to the swarm talker. It uses the Bluzelle API to talk to the swarm I have just created. First thing is I need to connect to the swarm using the Bluzelle API initialize function which takes a public and private key for the cryptography part of creating and reading keys. It also takes an endpoint so that it knows where the swarm is and a swarm ID. The endpoint is my local host address and the port of one of the nodes in the swarm.

· · 10	t main()	
	const std::string END_POINT{"ws://122760.0531:49150"};	
9.	if (brani::initialize(oub key, priv key, END POINT, SWARM ID))	
	(
2	try	
5	<pre>comst storrstring de_uutd(myde); stdr:shared_ptr<bzapi::database> cache(bzapi::has_db(db_uuid) ? bzapi::open_db(db_uuid)</bzapi::database></pre>	
7		
0	std:string response(cacheswarm status()):	
0	<pre>std::cout << response << std::endl;</pre>	
1		
	response = cache->create("key0", "A very mice value.");	
3	<pre>std::cout << "create:\n" << response << std::endl;</pre>	
ŝ	response = cache->quick read("key0"):	
6	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>	
7		
8	response = cache->update("key0", "A better value than the old one.");	
9	staticout << "update:\n" << response << statiend:	
0	response = cache->ouick read("kev@"):	
2	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>	
3		
4	response = cache->keys();	
	stallcout << "keysi\n" << response << stallend()	
μ		

It will ask that one node for a list of the rest of the nodes in the swarm and then it will ask each of those swarms for a status and choose the fastest node to connect to from then on. Every now and then it pulls the swarm to find out which one is fastest and always tries to use the fastest node. Once we've initialized that connection we can create or open a cache and that's what this line here is for:

if (bzapi::initialize(pub key, priv key, END POINT, SWARM ID))	
try	
const std::string db uuid{"mvdb"}:	
std::shared_ptr <bzabi::database> cache(_bzabi::has_db(db_uuid) ? bzabi::open_db(db_uuid)</bzabi::database>	
: bzapi::create_db(db_uuid));	
111111111111111111111111111111111111111	
<pre>std::string response(cache->swarm_status());</pre>	
<pre>std::cout << response << std::endl;</pre>	
and the second state and the second sec	
response = cache->create("Key8", "A very mice value.");	
<pre>std::Cout << "create:\n" << response << std::endl;</pre>	
and a second s	
response = Cache-quick_read("key0");	
storrout << "quick rearin" << response << storrengl;	
second a such a address of the second state of the state of the second state of the se	
response = cache-suppare(keye , 'A better value than the old one.);	
staricout << "update:\n" << response << stariend;	
recentre - such - such read/thements	
statismut a service - quark_cover any //	
station of quick reality of response of stationary	
response = carbe-skeys().	
stditiout of "kewile" of response of stditendit	
are to a set a	
response = cache->remove("kev8"):	
std::cout < "remove:\n" << response << std::end:	
	T

So I have chosen myDB as the identifier for my cache.

if (bzapi::initialize(pub_key, priv_key, END_POINT, SWARM_ID))	
try	
(I	
<pre>const std::string db_uuid("mydi");</pre>	
<pre>std::shared_ptr<bzapi::database> cache(bzapi::has_db(db_uuid) ? bzapi::open_db(db_uuid)</bzapi::database></pre>	
<pre>std::string response(cache->swarm status());</pre>	
<pre>std::cout << response << std::endl;</pre>	
response = cache->create("key0", "A very nice value.");	
<pre>std::cout << "create:\n" << response << std::endl;</pre>	
represent - rache sevict read/"key8");	
std::cout << "mails reactive" << std::cout </td <td></td>	
response = cache->update("key0", "A better value than the old one.");	
<pre>std::cout << "update:\n" << response << std::endl;</pre>	
second - and a solid second (the del)	
response = cache-squick_read(keye); std::cout or "wick reading" or response or std::codi:	
storrenet of quark respirate of storrenet,	
response = cache->keys();	
<pre>std::cout << "keys:\n" << response << std::endl;</pre>	
response = cache->remove("key8");	
<pre>std:cout << Tremove:\n" << response << std:renal;</pre>	

We will first ask the swarm does it have that cache.

A names -	Change in the second seco
28	if (bzapi::initialize(pub_key, priv_key, END_POINT, SNARM_ID))
21	(
22	try
2.3	const stduction dh unid/mudb");
25	stdirshared ptrebzapizidatabases cache(szapizibas sh(db uuld)? bzapiziopen db(db uuld)
26	: bzapi::create_db(db_uuid)};
27	
28	
29	<pre>sto::string response(cache=>swarm_status()); std::rout <= response <= std::rout</pre>
31	providence of response of accidence
32	response = cache->create("key0", "A very mice value.");
33	<pre>std::cout << "create:\n" << response << std::endl;</pre>
34	and a sector substance and the sector s
32	response = cache-squick_read('keye'); std:read('s cache-squick_read('keye');
37	Stations of date team in or response of stations,
38	response = cache->update("key0", "A better value than the old one.");
39	<pre>std::cout << "update:\n" << response << std::endl;</pre>
40	recense - cache - autok mand/"hand").
42	stdurout or "main and and or exponse of stdurond)
43	stations of days into a transition of stational
44	response = cache->keys();
45	<pre>std::cout << "keys:\n" << response << std::endl;</pre>
46	The second
25	<pre>response = cache=>remove(keye); ctd::could = "remove(keye); ctd::could = "remove(keye);</pre>
40	

If it does we will open the cache. If it does not it will create the cache.

	if	(bza	pi::initialize(pub key, priv key, END POINT, SWARM ID))
	{		
		try	
			const std::string db uuid{"mvdb"}:
•			<pre>std::shared_ptr<bzapi::database> cache{ bzapi::has_db(db_uuid) ? bzapi::preate_db(db_uuid)</bzapi::database></pre>
			<pre>std::string response(cache->swarm_status());</pre>
			<pre>std::cout << response << std::endl;</pre>
			<pre>response = cache->create("kev@", "A very nice value."):</pre>
			<pre>std::cout << "create:\n" << response << std::endl;</pre>
			response = cache->guick_read("key0");
			<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
			response = cache->update("kev@", "A better value than the old one."):
			<pre>std::cout << "update:\n" << response << std::endl;</pre>
			response = cache->quick read("key#"):
			<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
			response = cache->kevs():
			<pre>std::cout << "keys:\n" << response << std::endl;</pre>
			<pre>response = cache->remove("kev0");</pre>
			<pre>std::cout << "remove:\n" << response << std::endl;</pre>
			111111111111111111111111111111111111111

Return value of these two functions is a database object:

if (bza	pi::initialize(pub_key, priv_key, END_POINT, SWARM_ID))
try	
(<pre>const std::string db_uuid{"mydb"};</pre>
	<pre>std::shared_ptr<bzapi::database> cache(bzapi::has_db(db_uuld) 7 Bzapi::open_db(db_uuld)</bzapi::database></pre>
	<pre>std::string response(ache->warm_status()); std::cut <= response <= std::end();</pre>
	<pre>response = cache->create("key0", "A very nice value."); std::cout << "create:\n" << response << std::endl;</pre>
	<pre>response = cache->quick_read("key0"); std::cout << "quick read:\n" << response << std::endl;</pre>
	<pre>response = cache->update("key0", "A better value than the old one."); std::cout << "update:\n" << response << std::endl;</pre>
	response = cache⊸quick_read("key0"); std::cout << "quick read:\n" << response << std::endl;
	<pre>response = cache->keys(); std::cout << "keys:\n" << response << std::endl;</pre>
	response = cache->remove("key0"); std::cout << "remove:\m" << response << std::endl;

Then we run this and get the following output:

swarmspooler	swarmtalker -				
[2019-05-30 [2019-05-30 348 bytes wi	11:58:44.108681] 11:58:44.108692]	[0x000070000fb05000] [0x000070000fb05000]	[debug] [debug]	<pre>(db_impl.cpp:175) - Got response for message 5 (db_impl.cpp:180) - Ignoring db response for unknown or already</pre>	processed
[2019-05-30 [2019-05-30 [2019-05-30 [2019-05-30 [2019-05-30 [2019-05-30 [2019-05-30 [2019-05-30	11:58:44.121517] 11:58:44.121551] 11:58:44.122566] 11:58:44.122591] 11:58:44.123631] 11:58:44.123631] 11:58:44.123675]	[0x000700001b05000] [0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000]	[debug] [debug] [debug] [debug] [debug] [debug] [debug]	<pre>(db_impl.cpp:175) - Got response for message 6 (db_impl.cpp:120) - 1 of 3 response received (db_impl.cpp:125) - Got response for message 6 (db_impl.cpp:125) - Got 7 seponse received (db_impl.cpp:125) - Got response for message 6 (db_impl.cpp:126) - 3 of 3 responses received (db_impl.cpp:199) - Dome processing db response for message 6</pre>	
{ "result" }	: 1			1	t
(2019-05-30	11:58:44.123746]	[0x0000700001b05000]	[debug]	(library.cpp:61) - Events run: 77	

For the demo the first thing I will do once I have a cache, is ask for the swarm status and print that out:

1546-000	V. CM	Send July 15	
		: bzapi::create_db(db_uuid)};	
		<pre>std::string response(cache->swarm_status());</pre>	
		std::cout << response << std::endl;	
		response = cache->create("key0", "A very nice value.");	
		<pre>std::cout << "create:\n" << response << std::endl;</pre>	
		A second se	
		response = cacne->quick_read('keye');	
		storicout « "quack remain" « response « storienal;	
		reconce = cache_windate/"kau@" "& better value than the old one ").	
		std::cout << "undate:\n" << response << std::end):	
		response = cache->guick_read("key0");	
		<pre>std::cout << "quick read:\n" << response << std::endl;</pre>	
		response = cache->keys();	
		<pre>std::cout << "keys:\n" << response << std::endl;</pre>	
		response = cache->remove('keyw');	
		storrcout << remover(n << response << storrengt;	

Then I'll create a key value pair and key0, A very nice value:

main.cop - A Ca	Added late test -	
6	: bzapi::create_db(db_wid);	
8	std::cout << regonae << std::end;};	
2	response = cache-screate(" key@", "A very nice value."); std::cout << "create:\n" << response << std::endl;	
* 5 5	<pre>response = cache->quick_read("key8"); std::cout << "quick read:\n" << response << std::endl;</pre>	
5 9	response = cache->update("key0", "A better value than the old one."); std::cout << "update:\n" << response << std::endl;	
1	response = cache->quick_read("key8"); std::cout << "quick read:\n" << response << std::end\;	
4 5 5	response = cache->keys(); std::cout << "keys:\n" << response << std::endl;	
7 8 9	response = cache->remove("key0"); std::cout << "remove:\m" << response << std::endl; ////////////////////////////////////	

Then I'll quick read that key0:

	are the area and the area area area area area area area ar
	: bzap1::create_db(db_uuld)};
	<pre>std::string response(cache->swarm status());</pre>
	<pre>std::cout << response << std::endl;</pre>
	<pre>response = cache->create("key0", "A very nice value.");</pre>
	<pre>std::cout << "create:\n" << response << std::endl;</pre>
-	response = cache->quick_read("keye");
Accession in succession	<pre>std::cout << "quick read:\n" << response << std::enal;</pre>
	response = rache_pundate("key8", "A better value than the old one ").
	stdirout of "undatails" of percents of the and is
	response = cache->quick read("kev@");
	<pre>std::cout << "guick read:\n" << response << std::endl;</pre>
	response = cache->keys();
	<pre>std::cout << "keys:\n" << response << std::endl;</pre>
	response = cache->remove("key0");
	<pre>std::cout << "remove:\n" << response << std::endl;</pre>

Then I'll update the value of key0:

a mento	· A CMakeList	N.Iat =
18 19 10		<pre>std::string response(cache->swarm_status()); std::cout << response << std::end(;</pre>
32 33		<pre>response = cache->create("key0", "A very nice value."); std::cout << "create:\n" << response << std::endl;</pre>
35 36		response = cache->quick_read("key0"); std::cout << "quick read:\n" << response << std::endl;
38	•	<pre>response = cache->update("key0", "A better value than the old one."); std::cout << "update:\n" << response << std::endl;</pre>
10 11 12		<pre>response = cache->quick_read("key@"); std::cout << "quick read:\n" << response << std::endl;</pre>
13 14 15		<pre>response = cache->keys(); std::cout << "keys:\n" << response << std::endl;</pre>
17 18 19		<pre>response = cache->remove("keyd"); std::cout << "remove:\n" << response << std::endl;</pre>

Read it again to be convinced that the value has been updated:

1.0	and a second and a second seco
10	staticout << response << statienal;
14	rerease - cache - create ("key8" "& yery sice yalue ");
13	stdivenuit es "preates a se stdivendi"
14	store of createring of response of store and the
15	response = cache->quick read("kev8"):
36	<pre>std::cout << "guick read:\n" << response << std::endl;</pre>
17	
88	response = cache->update("key0", "A better value than the old one.");
19	<pre>std::cout << "update:\n" << response << std::endl;</pre>
18	
1	response = cache->quick_read("key0");
2	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
13	second - such - here().
15	response = cache-skeys();
16	stor.cout of keys in a response of stor.endt,
17	response = cache->remove("kev@"):
18	<pre>std::cout << "remove:\n" << response << std::endl:</pre>
19	111111111111111111111111111111111111111
50	
51	}
2	<pre>catch(const std::exception &e)</pre>
13	(
54	<pre>std::cout << "Caught exception: [" << e.what() << "]" << std::endl;</pre>

And then I'll ask for all the keys in the cache:

1.1	and a state of the
3.2	response = cache->create(keye, "A very nice value.");
3.6	sturrent of createrin of response of sturrent,
35	response = cache->quick read("kev0"):
36	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
37	
38	response = cache->update("key0", "A better value than the old one.");
39	<pre>std::cout << "update:\n" << response << std::endl;</pre>
40	reconness = cache_squick read("kau8")+
12	std::cout << "guick read:\n" << response << std::endl:
43	
14	response = cache->keys();
15	std::cout << "keys:\n" << response << std::endl;
4D 67	response = cache->remove("kav@"):
68	<pre>std::cout << "remove:\n" << response << std::endl;</pre>
69	11/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1/1
50	
51	}
24	aten(const storrexception de)
54	<pre>std::cout << "Caught exception: [" << e.what() << "]" << std::endl:</pre>
1.0	

Finally I will remove the key0 from the cache to clean up:

main.cpp /	A CMakeUsta.bt =
2	response = cache->create("key0", "A very nice value.");
3.	<pre>std::cout << "create:\n" << response << std::endl;</pre>
6	
5	<pre>response = cache->quick_read("key0");</pre>
5	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
5	response = cache->update("key0", "A better value than the old one.");
9	<pre>std::cout << "update:\n" << response << std::endl;</pre>
8	
1	<pre>response = cache->quick_read("key0");</pre>
2	<pre>std::cout << "quick read:\n" << response << std::endl;</pre>
3	
6	response = cache->keys();
2	<pre>std::cout << "keys:\n" << response << std::endl;</pre>
	(H)
	response = cache->remove('keye');
	sta::cout << "remove:\n" << response << sta::enal;
2	
2	
	entch(const stducection to)
6	(
	stdurout or "Caught exception: [" or a shat() or "]" or stdurond);
	storreout a cought exception 1 a country of 1 a storrengt,
6	,
e	

And if we look at the output you can see what happened. That is our status output which lists all of the nodes and their connection information:

1	
1	
	"host" : "127.0.0.1".
	"latency" : 4516.
	"port" ± 49158.
	"uu3d" * "MFYwEAYHKoZIzj@CAQYFK4EEAAoDQgAEb21xjv3dr09cbYyHum7wD08VdfKL5knuaTspWfYkvl2N3oVUVKU5JWaxaUEpj1/2004x2Mbhd6v6JQ@aG
},	
{	
	"host" : "127.0.0.1",
	"latency" : 0,
	"port" : 49152,
	"uuld" : "MFYwEAYHKoZIzj@CAQYFK4EEAAoDQgAEMg2LnYtqcudTFmDV6RGT6IGUvQ0jhYP/o8j42501qbeD5X5q++mU/L8L2ET81PJquUXwpM1TDD4kuwjC
1	
1	Phoret . 1177 8 8 10
	Hoters Action to A
	"nort" + 40151
	"uuid": "MFYwEAYHXoZIzi8CA0YFK4EEAAoD0aAE8xJvKKG4+k0T5aFL6T8M7ee8tLzCJkUYma8+bVC6WXupXa18PU4pLac7T8N2VXxmGY47Ad5hMaExxckR
1.	
(
	"host" : "127.0.0.1",
	"latency" : 0,
	"port" : 49153,
	"uuld" : "MFYwEAYHKOZIZJ@CAQYFK4EEAAoDQgAElLwWPBMwPRlCoK7Yxbb2TW58F8b3anPm834sulhibZeIB4hw/oG@YUTry5V9LGBTjl8WozHMr94YqOKD
,)	
1.	

And the time it took to talk to them:

{	
	"host" : "127.0.0.1",
	"latency" : 4510,
	"port" : 49150,
	"uuid" : "MFYwEAYHKoZIzj@CAQYFK4EEAAoDQgAEb21xjv3dr09cbYyHum7wD0BVdfKL5knuaTspWfYkvl2N3oVUVKUSJWaxaUEpj1/XXXx2Wbhd6v6J0@aGncBy70=="
}.	
(
	"host" : "127.0.0.1".
	"latency" : 0,
	"port" : 49152,
	"uuid" : "MFYwEAYHKoZIzj@CAQYFK4EEAAoDQgAEMg2LnYtqcudTFmDV6RGT6IGUvQ0jhYP/o8j4250iq6eD5X5q++mU/L8lZET8iPJquUXwpM1fDD4kuwg06mV3AA=="
}.	I
{	
	"host" : "127.0.0.1",
	"latency" : 0,
	"port" : 49151,
	"uuid" : "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE8xJyKKG4+kQT5qFL6T8M7ee0tLzCJkUYmqB+bVC6WXupXa18PU4pLgc7T8N2VXxmGY47AdShMgExxckRN55wKA=="
},	
{	
	"host" : "127.0.0.1",
	"latancu" · A

It also gives us the value of the primary node:

"uuid" : "MFYwEA"	YHKoZIzj@CAQYFK4EEAAo	QgAE1LwW	PBMwPR1CoK7Yxbb2TW5BF8b3anPm834sulhibZeIB4hw/oG0YUTry5V9LGBTjl8WozH
} primary_node" : "Warm_dA "swarm_git_commit" : "0. "swarm_version" : "", "uptime" : "0 days, 0 hou }	YHKO213500AQY3X455AA03 3.1096-139-gc5da29c-d urs, 2 minutes"	DOgAE8xJy irty",	
[2019-05-30 11:58:44.047947] 3 bytes written 3 bytes written 3 bytes written 360 bytes written	[0x00000010c8db5c0]	[debug]	(db_impl.cpp:81) - Sending database request for message 1
500 09/05 #71100 [2019-05-30 11:58:44.064882] [2019-05-30 11:58:44.064943] [2019-05-30 11:58:44.066063] [2019-05-30 11:58:44.066547] [2019-05-30 11:58:44.067581] [2019-05-30 11:58:44.067581]	[0x800070000fb65000] [0x800070000fb65000] [0x800070000fb65000] [0x800070000fb65000] [0x800070000fb65000] [0x800070000fb65000] [0x800070000fb65000]	[debug] [debug] [debug] [debug] [debug] [debug]	(db_ispl.cpp:175) - foot response for message 1 (db_ispl.cpp:128) - 1 of 3 responses received (db_ispl.cpp:175) - foot response for message 1 (db_ispl.cpp:126) - 2 of 3 responses received (db_ispl.cpp:175) - foot response for message 1 (db_ispl.cpp:128) - 3 of 3 responses received
<pre>[2019-05-30 11:58:44.06/039] create: { "result" : 1 }</pre>	[6x8866,6666,062062	[debug]	(db_impl.cpp:139) - Done processing do response for message 1
[2019-05-30 11:58:44.067737] [2019-05-30 11:58:44.068724] [2019-05-30 11:58:44.068739]	[0x000000010c8db5c0] [0x000070000fb05000] [0x000070000fb05000]	[debug] [debug] [debug]	<pre>(db_impl.cpp:81) - Sending database request for message 2 (db_impl.cpp:175) - Got response for message 1 (db_impl.cpp:180) - Ignoring db response for unknown or already p</pre>

Here is the create response we see that it was successful:

2019-05-30 11:58:44.064943) 2019-05-30 11:58:44.066063] 2019-05-30 11:58:44.066096]	[0x000070000fb05000] [0x000070000fb05000] [0x000070000fb05000]	[debug] [debug]	<pre>(db_impl.cpp:210) - 1 of 3 responses received (db_impl.cpp:175) - Got response for message 1 (db_impl.cpp:210) - 2 of 3 responses received</pre>
[2019-05-30 11:58:44.067547] [2019-05-30 11:58:44.067581] [2019-05-30 11:58:44.067639]	[0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000]	[debug] [debug] [debug]	<pre>(db_impl.cpp:175) - Got response for message 1 (db_impl.cpp:210) - 3 of 3 responses received (db impl.cpp:199) - Done processing db response for message 1</pre>
reate: "result" : 1			
1			
2019-05-30 11:58:44.067737] 2019-05-30 11:58:44.068724] 2019-05-30 11:58:44.068739] 2019-05-30 11:58:44.069758] 2019-05-30 11:58:44.069772] 47 bytes written	[0x00000010c8db5c8] [0x000700001b05000] [0x000700001b05000] [0x000700001b05000] [0x000700001b05000] [0x000700001b05000]	[debug] [debug] [debug] [debug] [debug]	(db_impl.cpp:31) - Sending database request for message 2 (db_impl.cpp:175) - Got response for message 1 (db_impl.cpp:186) - Ignoring do response for indinoum or already process (db_impl.cpp:175) - Got response for message 1 (db_impl.cpp:186) - Ignoring do response for indinoum or already process
2019-05-30 11:58:44.069870] 2019-05-30 11:58:44.069877] 2019-05-30 11:58:44.069931] 2019-05-30 11:58:44.069931] 2019-05-30 11:58:44.069931]	[0x000070000fb05000] [0x000070000fb05000] [0x000070000fb05000]	[debug] [debug] [debug]	<pre>(db_impl.cpp:175) - Got response for message 2 (db_impl.cpp:210) - 1 of 1 responses received (db_impl.cpp:199) - Done processing db response for message 2</pre>
"key" : "key0", "result" : 1, "value" : "A very nice va	lue."		

If it were not successful, it would also give us an error message, for example, the key already exists. Here's the result of the quick read. So I asked for key0 and it gave me the value of a very nice value:

2019-05-30 11:58:44.068/24]	[888530198881988819881]	[debug]	(db_impl.cpp:1/5) - Got response for message 1	
2019-05-30 11:58:44.008/39]	[0x0000700001D05000]	[debug]	(db_impl.cpp:180) - ignoring db response for unknown or already pr	ocessed
2019-05-30 11:50:44.009/30]	[0x0000700001005000] [0x0000700001005000]	[debug]	(db_imp] cop:180) - Toporing db response for unknown or already or	neassed
17 hytes written	[0x0000700001003000]	(debug)	(ub_tip(tepp.too) - ignoring up response for unknown or accessly pr	ocessed
2019-05-30 11:58:44,269870]	[8x8888788881b85888]	[debug]	(db impl.cop:175) - Got response for message 2	
2019-05-30 11:58:44.069897]	[0x000070000fb05000]	[debug]	(db impl.cpp:210) - 1 of 1 responses received	
2019-05-30 11:58:44.0699311	[exeeee7aeee1be5aee]	(debug)	(db_impl.cpp:199) - Done processing db response for message 2	
"key" : "key0",				
"result" : 1,				
value i k very nice vi	0106.			
				1
2019-05-30 11:58:44.070015]	[0x000000010c8db5c0]	[debug]	(db_impl.cpp:81) - Sending database request for message 3	
82 bytes written				
2019-05-30 11:58:44.0852891	[0x000070000fb05000]	[debug]	<pre>(db_impl.cpp:175) - Got response for message 3</pre>	
ters to so warser		[dobug]	(db incl ency 310) 1 of 3 conserves merelyed	
2019-05-30 11:58:44.085325]	[0x000700001b05000]	[debug]	(db_imp(.cpp:210) - 1 of 3 responses received	
2019-05-30 11:58:44.085325] 2019-05-30 11:58:44.086335]	[0x0000700001505000] [0x0000700001505000]	[debug]	(db_impl.cpp:175) - Got response for message 3	
2019-05-30 11:58:44.085325] 2019-05-30 11:58:44.086335] 2019-05-30 11:58:44.086361]	[0x0000700001b05000] [0x0000700001b05000] [0x0000700001b05000]	[debug] [debug]	<pre>(db_impl.cpp:210) - 1 of 3 response freceived (db_impl.cpp:175) - 60t response for message 3 (db_impl.cpp:210) - 2 of 3 responses received</pre>	
2019-05-30 11:58:44.085325 2019-05-30 11:58:44.086335 2019-05-30 11:58:44.086335 2019-05-30 11:58:44.086361 2019-05-30 11:58:44.087424	[0x000700001b05000] [0x000700001b05000] [0x000700001b05000] [0x0000700001b05000]	[debug] [debug] [debug]	<pre>(db_impl.cpp:175) - Got responses received (db_impl.cpp:175) - Got responses for message 3 (db_impl.cpp:175) - Got responses received (db_impl.cpp:175) - Got response for message 3</pre>	
2019-05-30 11:58:44.085325 2019-05-30 11:58:44.086335 2019-05-30 11:58:44.086335 2019-05-30 11:58:44.086361 2019-05-30 11:58:44.087424 2019-05-30 11:58:44.087455	[0x8000700001b05000] [0x8000700001b05000] [0x8000700001b05000] [0x8000700001b05000] [0x8000700001b05000] [0x8000700001b05000]	[debug] [debug] [debug] [debug]	<pre>(db_impl.cpp:210) - 1 of 3 responses received (db_impl.cpp:175) - Got response for message 3 (db_impl.cpp:120) - 2 of 3 responses received (db_impl.cpp:175) - Got response for message 3 (db_impl.cpp:210) - 3 of 3 responses received</pre>	
2019-05-30 11:58:44.085325] 2019-05-30 11:58:44.086335] 2019-05-30 11:58:44.086361] 2019-05-30 11:58:44.087424] 2019-05-30 11:58:44.087455] 2019-05-30 11:58:44.087500]	[0x0000700007b05000] [0x000700007b05000] [0x0000700007b05000] [0x0000700007b05000] [0x000070000fb05000] [0x000070000fb05000] [0x000070000fb05000]	[debug] [debug] [debug] [debug] [debug]	<pre>(Bb_impl.cps:/zw) - 1 of 3 responses received (Bb_impl.cps:/IS) - 60t response for message 3 (Bb_impl.cps:/IS) - 60t response received (Bb_impl.cps:/IS) - 60t response received (Bb_impl.cps:/IB) - 3 of 3 responses received (Bb_impl.cps:/IB) - Dome processing Bb response for message 3</pre>	
2019-05-30 11:58:44.085325] 2019-05-30 11:58:44.085335] 2019-05-30 11:58:44.086361] 2019-05-30 11:58:44.0875424] 2019-05-30 11:58:44.087455] 2019-05-30 11:58:44.087500] pdate:	[0x0000700007b05000] [0x000700007b05000] [0x000700007b05000] [0x0000700007b05000] [0x000070000fb05000] [0x000070000fb05000] [0x000070000fb05000]	[debug] [debug] [debug] [debug] [debug]	<pre>(db_impl.cpp:12) - for 3 responses received (db_impl.cpp:13) - for response for necessage 3 (db_impl.cpp:13) - for response received (db_impl.cpp:13) - for response received (db_impl.cpp:14) - 3 of 3 responses received (db_impl.cpp:19) - Done processing db response for message 3</pre>	
2019-05-30 11:58:44.0853251 2019-05-30 11:58:44.0863351 2019-05-30 11:58:44.086361 2019-05-30 11:58:44.087451 2019-05-30 11:58:44.0874551 2019-05-30 11:58:44.0875500] pdate: "result" : 1	[0x8000700007b85000] [0x8000700007b85000] [0x8000700007b85000] [0x8000700007b85000] [0x8000700007b85000] [0x8000700007b85000] [0x8000700007b85000]	[debug] [debug] [debug] [debug] [debug]	<pre>(db_impl.cpp:/240 - 1 of 3 responses received (db_impl.cpp:/215) - 60 response to received (db_impl.cpp:/210 - 2 of 3 responses received (db_impl.cpp:/210 - 3 of 3 responses received (db_impl.cpp:/210 - 3 of 3 responses received (db_impl.cpp:/210 - 3 of 3 responses received)</pre>	

I then did the update or I changed a very nice value for key0 to something else:

"value" : "A very nice va }	alue."		
[2019-05-30 11:58:44.070015]	[0x000000010c8db5c0]	[debug]	(db_impl.cpp:81) - Sending database request for message 3
2019-05-30 11:58:44.085289]	[ex000070000fb05000]	[debug]	(db impl.cpp:175) - Got response for message 3
[2019-05-30 11:58:44.085325]	[0x000070000fb05000]	[debug]	(db_impl.cpp:210) - 1 of 3 responses received
[2019-05-30 11:58:44.086335]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 3
[2019-05-30 11:58:44.086361]	[0×000070000fb05000]	[debug]	(db_impl.cpp:210) - 2 of 3 responses received
[2019-05-30 11:58:44.087424]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 3
[2019-05-30 11:58:44.087455]	[ex0000700001505000]	[debug]	(db_impl.cpp:210) - 3 of 3 responses received
[2019-05-30 11:58:44.087500]	[0x000070000fb05000]	[debug]	<pre>(db_impl.cpp:199) - Done processing db response for message 3</pre>
"result" : 1			
[2019-05-30 11:58:44.087594]	[0x000000010c8db5c0]	[debug]	(db_impl.cpp:81) - Sending database request for message 4
[2019-05-30 11:58:44.088678]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 3
[2019-05-30 11:58:44.088693]	[0x000070000fb05000]	[debug]	(db_impl.cpp:180) - Ignoring db response for unknown or already processed
[2019-05-30 11:58:44.089775]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 3
[2019-05-30 11:58:44.089791]	[0x000070000fb05000]	[debug]	<pre>(db_impl.cpp:180) - Ignoring db response for unknown or already processed</pre>
348 bytes written			
[2019-05-30 11:58:44.089895]	[0x0000700201b05000]	[debug]	(db_impl.cpp:175) - Got response for message 4
	[0x0000700001505000]	[debug]	(db_impl.cpp:210) - 1 of 1 responses received
[2019-05-30 11:58:44.089958]	[execes/eeespeses]	[geoug]	(db_imp(.cpp:iaa) - bone processing db response for message 4
uick redu:			
"key" : "key8".			
"result" . 1			

And here it is a better value than the old one:

2019-05-30 11:58:44.088678]	[0x000000010c8db5c0] [0x000070000fb05000]	[debug]	<pre>(db_impl.cpp:81) - Sending database request for message 4 (db_impl.cpp:175) - Got response for message 3</pre>	
2019-05-30 11:58:44.088693]	[0x0000700001b05000]	[debug]	<pre>(db_impl.cpp:180) - Ignoring db response for unknown or already pr (db_impl.cpp:175) - Got response for message 3</pre>	ocessed 1
2019-05-30 11:58:44.089791]	[0x0000700001b05000]	[debug]	(db_impl.cpp:180) - Ignoring db response for unknown or already pr	ocessed i
48 bytes written 2019-05-30 11:58:44.0898951	[ex0000700001b05000]	[debug]	(db impl.cpp:175) - Got response for message 4	
2019-05-30 11:58:44.009923]	[0x0000700001b05000]	[debug]	(db_impl.cpp:210) - 1 of 1 responses received	
2019-05-30 11:50:44.009950] uick read:	[0x000700001505000]	(debug)	(db_impl.cpp:199) - Done processing db response for message 4	
"key" : "key@",				
"result" : 1, "value" : "A better valu	than the old one."			
Statement of the second second	Constitution and a state			
2019-05-30 11:58:44.090066]	[0x000000010c8db5c0]	(debug)	(db_impl.cpp:81) - Sending database request for message 5	
40 bytes written	[0-00070000fb05000]	[debug]	(db imp) const175) - Got corpores for mercane 5	
2019-05-30 11:58:44.1044361	[0x000070000fb05000]	[debug]	(db impl.cop:210) - 1 of 3 responses received	
2019-05-30 11:58:44.105510]	[0x0000700001b05000]	[debug]	(db_impl.cpp:175) - Got response for message 5	
2019-05-30 11:58:44.105539]	[0x000070000fb05000]	[debug]	(db_impl.cpp:210) - 2 of 3 responses received	
2019-05-30 11:58:44.106519]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 5	
2019-05-30 11:58:44.106543]	[0x000070000fb05000]	[debug]	(db_impl.cpp:210) - 3 of 3 responses received	
2019-05-30 11:58:44.106643]	[0x0000700001505000]	[debug]	(db_impl.cpp:199) - Done processing db response for message 5	
eys:				
eys: "keys" :				
eys: "keys" : [

Finally I asked the swarm for all the keys in our cache. There is only one so it returned an array of one key i.e. key0:

2019-05-30 11:58:44.106519]	[0x0000700001b05000]	[debug]	(db_impl.cpp:175) - Got response for message 5
2019-05-30 11:58:44.106543]	[0x0000700001b05000]	[debug]	(db_impl.cpp:210) - 3 of 3 responses received
2019-05-30 11:58:44.106643]	[0x000070000fb05000]	[debug]	(db_impl.cpp:199) - Done processing db response for message 5
keys:			
"keys" :			
rkeye	I		
1			
*			
2010-05-30 11+58+44 1067441	[0x000000010c8db5c0]	[debug]	(db (mp], cop:81) - Sending database request for message 6
2019-05-30 11:58:44, 1076631	[0x0000700001b05000]	[debug]	(db impl.cop:175) - Got response for message 5
2019-05-30 11:58:44, 1076751	[0x000070000fb05000]	[debug]	(db impl.cop:180) - Innoring db response for unknown or already processed
2019-05-30 11:58:44,108681]	[0x000070000fb05000]	[debug]	(db impl.cop:175) - Got response for message 5
2019-05-30 11:58:44.108692]	[0x000070000fb05000]	[debug]	(db impl.cpp:180) - Ignoring db response for unknown or already processed
348 bytes written			
2019-05-30 11:58:44.121517]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 6
2019-05-30 11:58:44.121551]	[0x000070000fb05000]	[debug]	(db_impl.cpp:210) - 1 of 3 responses received
2019-05-30 11:58:44.122566]	[0x00070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 6
[2019-05-30 11:58:44.122591]	[0x000070000fb05000]	[debug]	(db_impl.cpp:210) - 2 of 3 responses received
[2019-05-30 11:58:44.123601]	[0x0000700001b05000]	[debug]	(db_impl.cpp:175) - Got response for message 6
2019-05-30 11:58:44.1236311	[0x000070000fb05000]	[debug]	(db impl.cop:210) - 3 of 3 responses received

And finally I removed the key to clean up:

(2013-03-30 11.30.44.100043) kevs:	[evocee.eeo.neoo.neoone]	[nennā]	(nnTuhrichbitas) - nous bioresstuñ an reshouse ini messañe s
{			
"keys" :			
1			
"key0"			
, 1			
r -			
[2019-05-30 11:58:44.106744]	[0x000000010c8db5c0]	[debug]	(db impl.cpp:81) - Sending database request for message 6
[2019-05-30 11:58:44.107663]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 5
[2019-05-30 11:58:44.107675]	[0x0000700001b05000]	[debug]	(db_impl.cpp:180) - Ignoring db response for unknown or already processe
[2019-05-30 11:58:44.108681]	[0x000070000fb05000]	[debug]	(db_impl.cpp:175) - Got response for message 5
[2019-05-30 11:58:44.108692]	[0x0000700001b05000]	[debug]	<pre>(db_impl.cpp:180) - Ignoring db response for unknown or already processe</pre>
348 bytes written			
[2019-05-30 11:58:44.121517]	[0x0000700001b05000]	[debug]	(db_impl.cpp:175) - Got response for message 6
[2019-05-30 11:58:44.121551]	[0x0000700001505000]	[debug]	(db_impl.cpp:210) - 1 of 3 responses received
[2019-05-30 11:58:44.122506]	[0x0000700001b05000]	[debug]	(db_impl.cpp:1/5) - Got response for message 6
[2010-05-30 11:50:44:122351]	[0x0000700001005000] [0x0000700004b05000]	[debug]	(db_impl_cop:175) - Cot cereorse for message 6
[2019-05-30 11:58:44.123631]	[exaper7aper1665666]	[debug]	(db_impl.cop:218) = 3 of 3 responses received
[2019-05-30 11:58:44,123675]	[0x0000700001b05000]	[debug]	(db impl.cop:199) - Done processing db response for message 6
renove:	and succession of the		
(
"result" : 1			
•			
(3010 of 30 11.70.11 1337451		14.4	(1) (have an etc) . E water way 27
[2019-05-30 11:58:44.123746]	[exeee/eee106001062066]	[debug]	(library.cpp:61) - Events run: 77

Future Work

Bluzelle is not the only player in this game. It has competitors such as MaidSafeCoin, Siacoin, Storj and Filecoin. It is a certainty that one of these come out on top in the near future and investors in the BLZ token are betting that when the dust settles Bluzelle will be the last man standing.

Bluzelle has made sure that it is well-positioned to dominate the market by establishing offices in the west in Vancouver, and in the east in Singapore. It has also made alliances to help it capture market share and become the dominant decentralized database storage solution.

Currently most of its features are in beta phase while some are being scrutinised before being made available. When fully operational its storage capacity will be ten million times that of current corporate IT storage at 0.001 the cost per Tb of storage when compared with current costs. To put it in a nutshell, bluzelle promises all round data security.

Conclusion

Bluzelle is very easy to use and involves writing minimal code to cache our data to it. It has servers available in many data centers across multiple region thereby providing low latencies and high throughputs to applications. Tiering ensures that Bluzelle Cache can match our I/O performance needs. Data within Bluzelle Cache is globally replicated across more than 25 global regions. With its client libraries, one can easily and in a scalable manner store his/her data across bluzelle's servers. The user's data is always available, irrespective of faults, disasters, or attack vectors. There is no overhead for infrastructure deployment as there is nothing new to deploy. Customers only pay for the services they use. Customers are able to use Bluzelle as their primary data store or as a replica-set, depending on how they use bluzelle APIs in their codebase. Bluzelle provides all the coding patterns from which customers can choose. Also there are provisions for quickly adding additional servers to new data centers. Bluzelle uses both symmetric and asymmetric key cryptography to ensure that data can only be modified by its authorized owners/writers. No organization, including Bluzelle, has any mechanism to remove or modify data stored on Bluzelle. This is a level of security that simply cannot exist on a centralized platform including the cloud. It also means that data stored on Bluzelle belongs to and can only be affected by the owner of that data and no one else. Only they can remove it. Furthermore, Bluzelle will also be offering optional symmetric key encryption for data in transit and at rest, to ensure that only authorized parties can read the data.

References

- 1. Shubham Sharma., et al. "Detecting Insider Attacks on Databases using Blockchains". In proceedings of Workshop on Blockchain Technologies and its Applications, ISRDC, IIT Bombay (2017).
- 2. Weizhi Meng., et al. "When Intrusion Detection Meets Blockchain Technology: A Review". IEEE Access 6 (2018): 10179-10188.
- 3. Deepak K Tosh., et al. "Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack". 2017 17th IEEE/ ACM International Symposium on Cluster, Cloud and Grid Computing (2017).
- Tzvi Chumash and Danfeng Yao. "Detection and prevention of insider threats in database driven web services". IFIPTM 2009: Trust Management III (2009): 117-132.