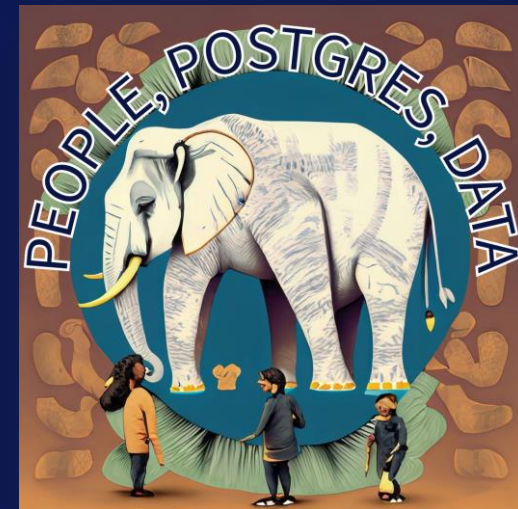


PERCONA

Cloud-Native PostgreSQL

Running PostgreSQL on Kubernetes

April 17, 2024
Peter Zaitsev,
Founder, Percona



Big Picture



Cloud

**Proprietary
Clouds bring
Great Usability
at Great Cost**

Have We
Been Here
Before?

2000s



2020s



Operating
Systems



Open Source Catches Up Again



- **Lock-in with Cloud Vendor**
- **Use Proprietary Solutions**
- **Highly Differentiated Cloud**
-
-



**CLOUD NATIVE
COMPUTING FOUNDATION**

- **Freedom to Run Anywhere**
- **Use Open Source**
- **Cloud Is Commodity**
- **Customer**
- **Choice of Vendors**



Giving Cloud its Originally Intended Role of Commodity Infrastructure

What is Cloud Computing?

An analogy: think of electricity services...

You simply plug into a vast electrical grid managed by experts to get a low cost, reliable power supply – available to you with much greater efficiency than you could generate on your own.

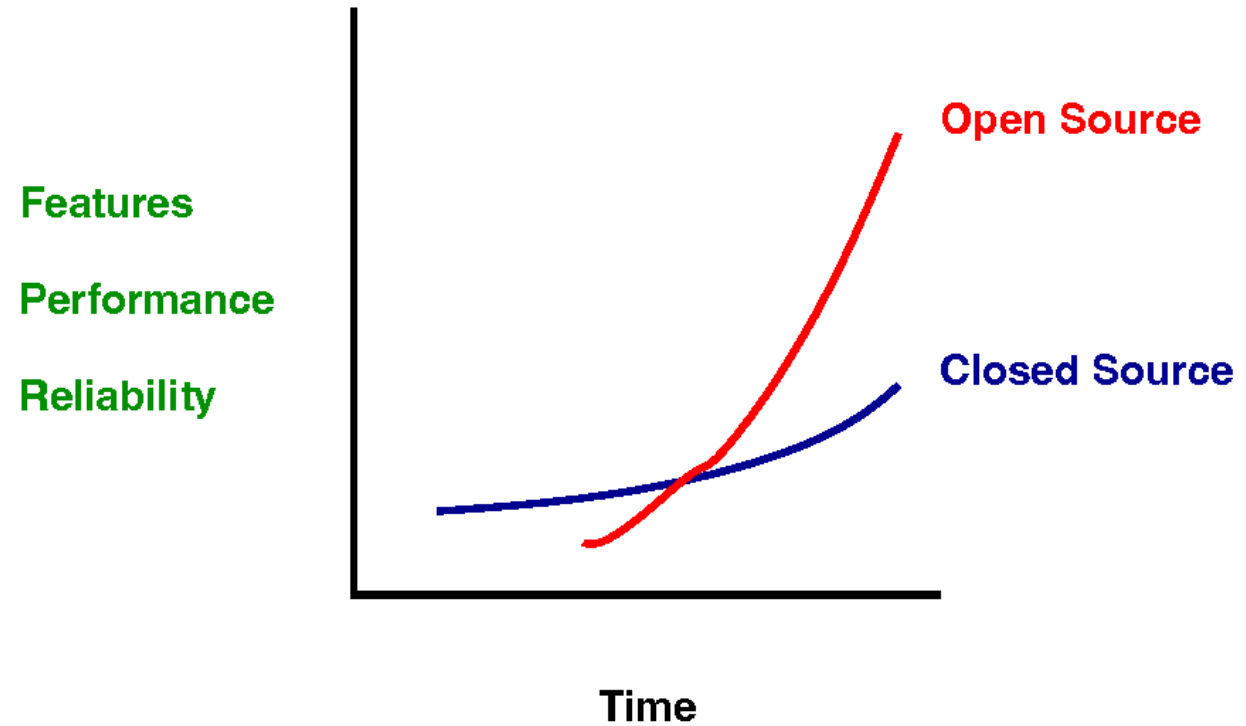


Power is a utility service - available to you on-demand and you pay only for what you use.



Open Source and Proprietary

Rise of Open Source



12/51

<https://momjian.us/main/writings/pgsql/forever.pdf>

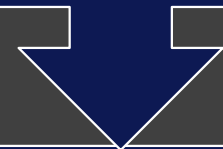
Kubernetes



Kubernetes is universally available



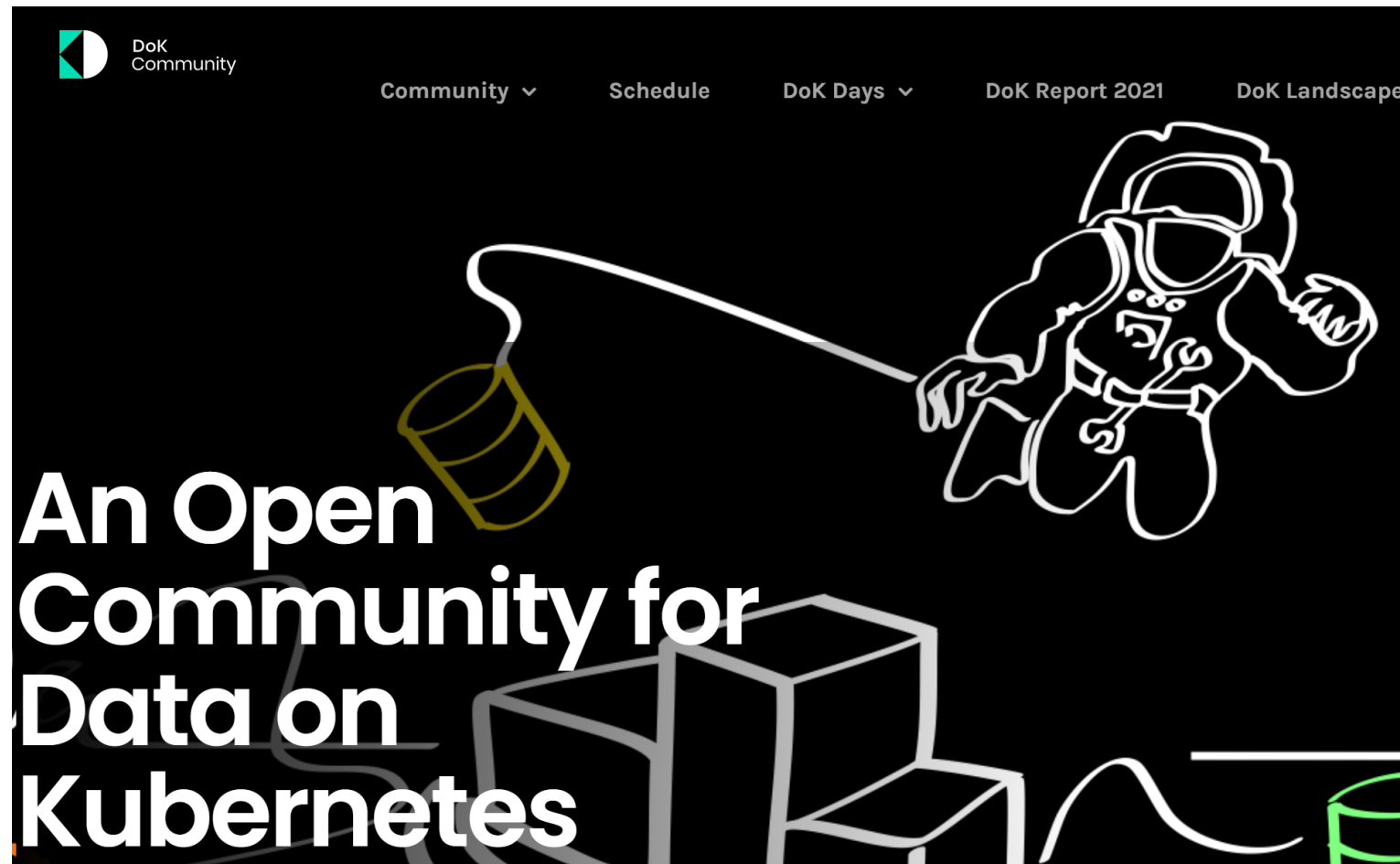
Kubernetes is getting better for stateful applications



Kubernetes Operators are available for most popular Open Source Databases

kubernetes

Data on Kubernetes



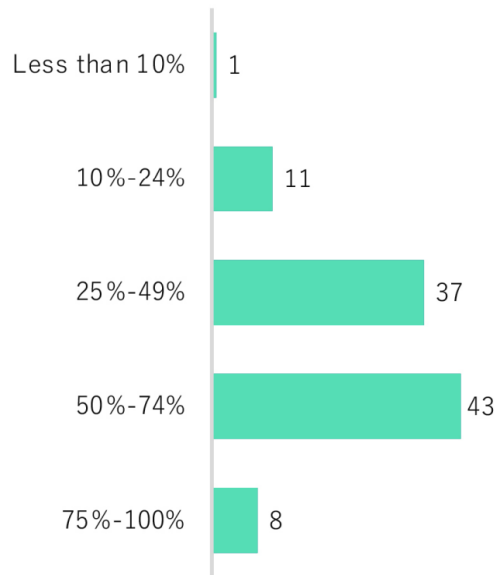
<https://dok.community/>

Data on Kubernetes

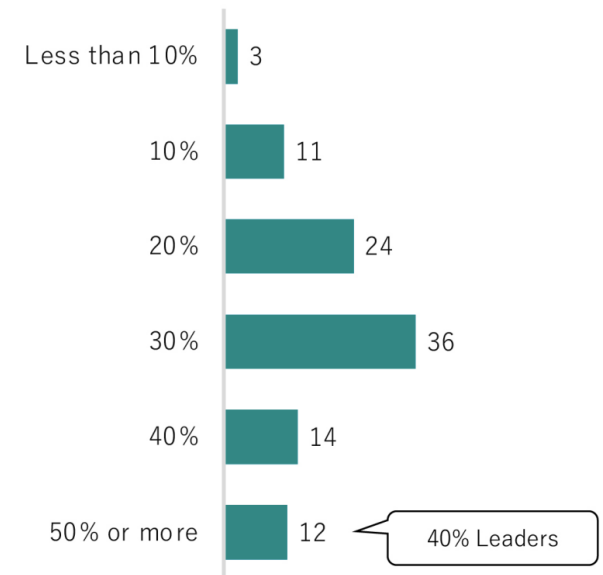
DoK workload %s are already high, and expected to increase

Leaders are chomping at the DoK bit

% of data workloads on k8s



Expected increase in data workloads on k8s

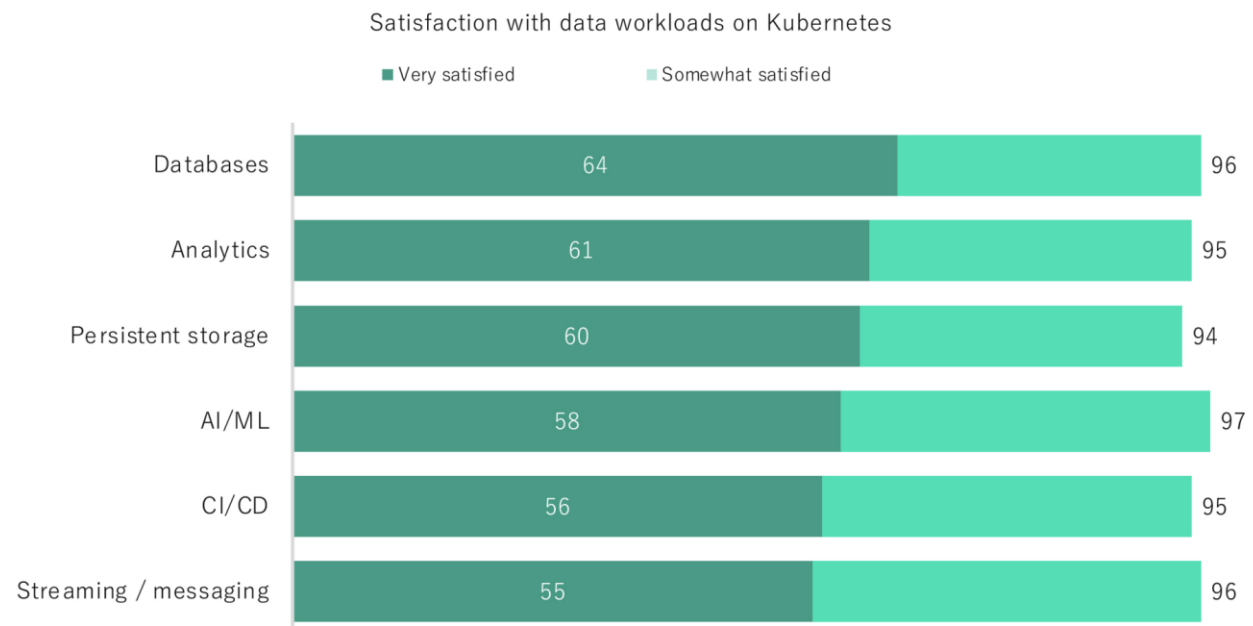


https://dok.community/wp-content/uploads/2022/10/DoK_Report_2022.pdf

And Happy
about That...

Satisfaction = Reality - Expectations

DoK is winning the expectations battle



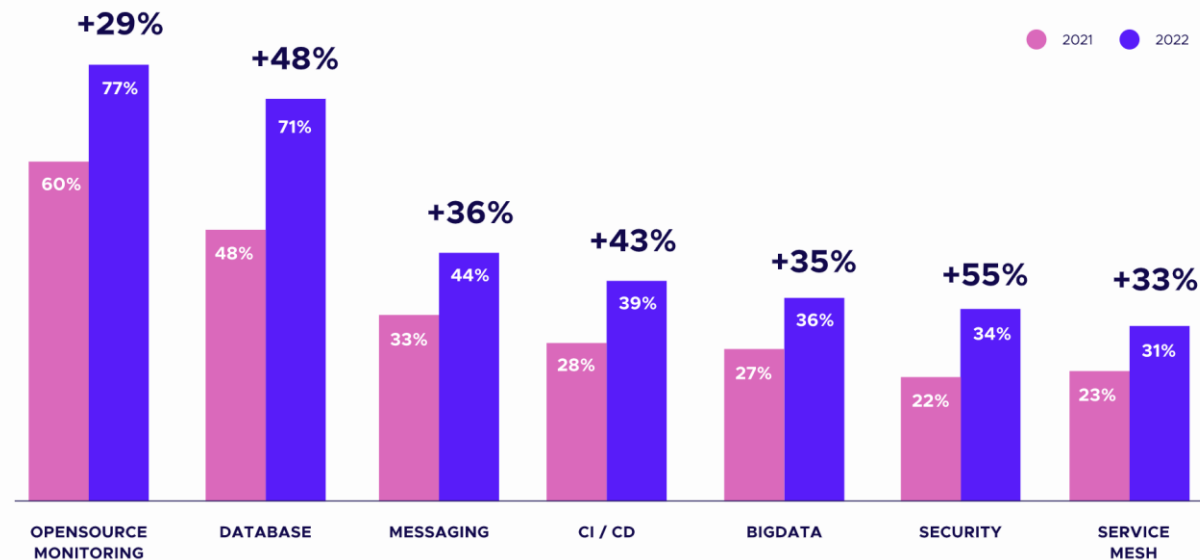
0. In general, how satisfied are you with using Kubernetes to run each of the following data workloads in your organization? Use a scale from 1 to 5 where 5 means "very satisfied" and 1 means "not at all"

Quite a growth in Adoption!

KUBERNETES GROWTH AREAS

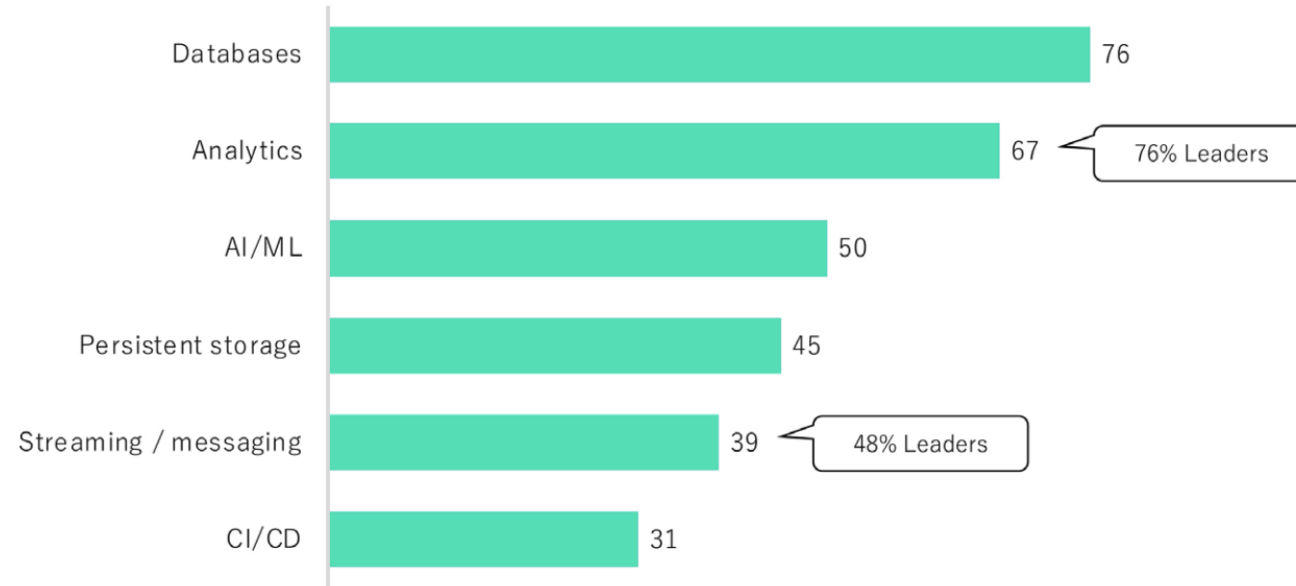
Focusing on non-application workloads, enterprises used an increasing variety of technologies. This reflects the need to enhance Kubernetes with better observability, security, and service-to-service communications. Other technologies enable specific use cases like CI/CD tools or databases.

Across all categories, **open source projects rank among the most frequently used solutions.**



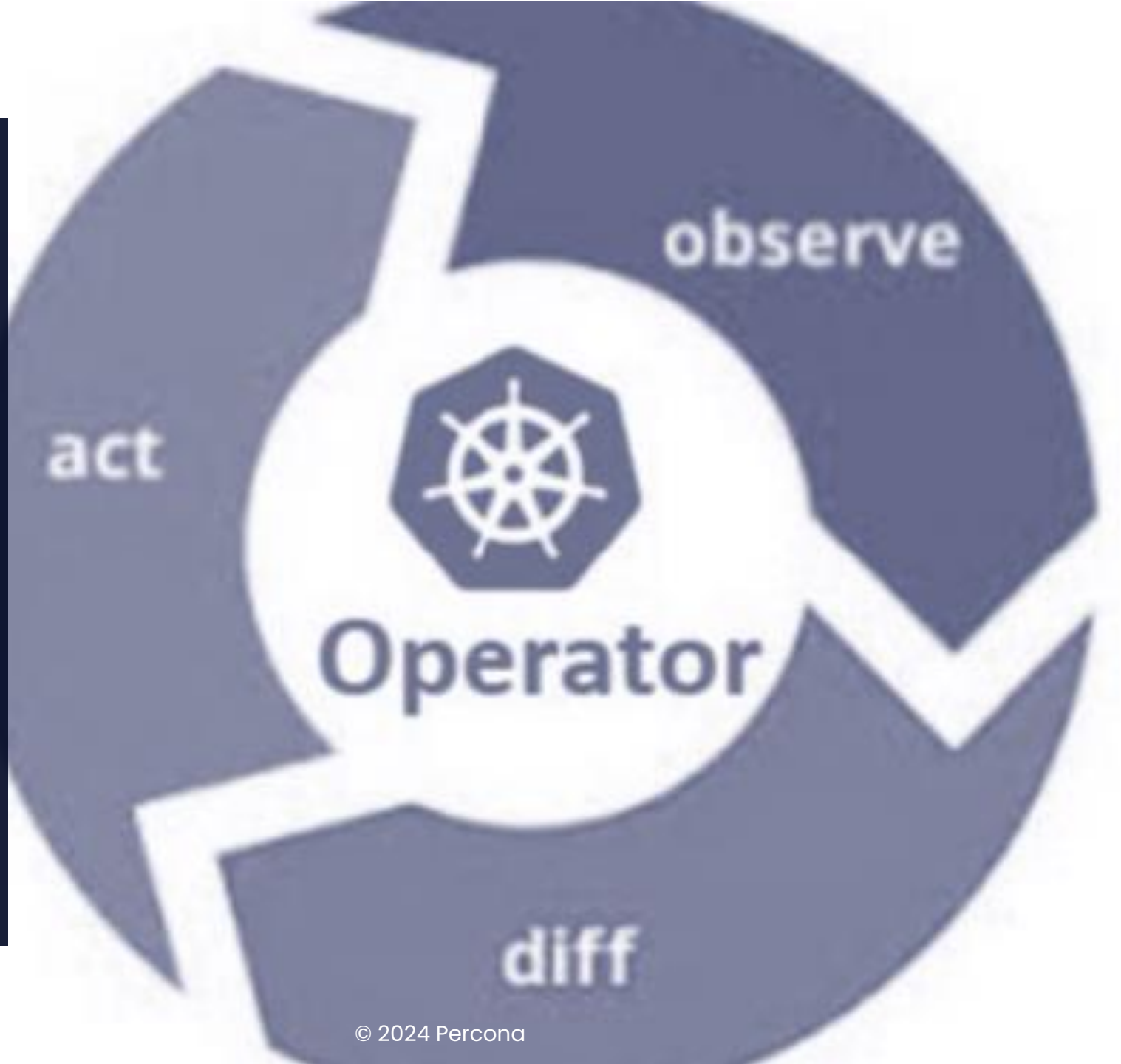
<https://www.cncf.io/reports/cncf-annual-survey-2022/>

Which data workloads on k8s



Databases on Kubernetes

Power of Kubernetes Operators



Power of Kubernetes Operators



- **Day One Automation**
 - Installation and Initial Configuration
- **Day Two Automation**
 - Backups, Scaling, Self Healing, Upgrades

Many Modern
DBaaS Are
Built on
Kubernetes
Operators



PostgreSQL Choices

CloudNativePG (EDB)

Zalando PostgreSQL Operator

Crunchy Data PostgreSQL Operator

Stackgres

Percona Operator for PostgreSQL

<https://www.percona.com/blog/run-postgresql-in-kubernetes-solutions-pros-and-cons/>

CloudNativePG

- **Built by EDB, Spun out as separate Project**
- **Apache 2.0**
- **Very Robust Feature Set**
- **Fantastic Documentation**
- **Bootstrap Cluster from Backup**
- **Not using Stateful Sets or Patroni**

Zalando PostgreSQL Operator

- **The Oldest PostgreSQL Operator**
- **MIT**
- **No Commercial Support from Creator**
- **Relies on Spilo (Docker PostgreSQL + Patroni)**
- **Extensive AWS EKS Support, like Volume Resize**

Zalando Gui

New PostgreSQL cluster

Cluster YAML definition

```
kind: "postgresql"
apiVersion: "acid.zalan.do/v1"

metadata:
  name: "acid-"
  namespace: "default"
  labels:
    team: acid

spec:
  teamId: "acid"
  postgresql:
    version: "11"
    numberOfInstances: 2
    enableMasterLoadBalancer: true
  volume:
    size: "10Gi"
  users:
    app_user: []
  databases:
    app_db: app_user
  allowedSourceRanges:
    # IP ranges to access your cluster go here

resources:
  requests:
    cpu: 100m
    memory: 1Gi
  limits:
    cpu: 1000m
    memory: 1Gi
```

New cluster configuration

Validate Copy definition Create cluster

Name	<input type="text" value="new-cluster (can be 48 characters long)"/>			
Namespace	<input type="text" value="default"/>			
Owning team	<input type="text" value="acid"/>			
PostgreSQL version	<input type="text" value="11"/>			
DNS name:	acid-.default			
Number of instances	<input type="text" value="2"/>			
Master load balancer	<input checked="" type="checkbox"/> Enable master ELB			
Replica load balancer	<input type="checkbox"/> Enable replica ELB			
Volume size	<input type="text" value="10"/> <input type="text" value="Gi"/>			
+ Users	<input type="text" value="app_user"/>			
+ Databases	<input type="text" value="app_db"/> <input type="text" value="app_user"/>			
Resources	CPU	Request	100	m
		Limit	1000	m
	Memory	Request	1	Gi
		Limit	1	Gi

Crunchy Data PostgreSQL Operator

- **Second to Market after Zalando Operator**
- **Apache 2.0 for Operator but Subscription Needed for PostgreSQL Images**
- **Sync and Async Replication Support**
- **Extension and User Management through Kubernetes**
- **Additional components like TimescaleDB and pgAdmin 4**

Stackgres

- **UI or CRD For Every Functionality**
- **AGPLv3 for OSS + Proprietary Enterprise Version**
- **Babelfish, TimescaleDB, Citus, Supabase extension**
- **Envoy Proxy for Observability**
- **Cool features like Benchmarking**
- **Implemented in Java**

Stackgres UI

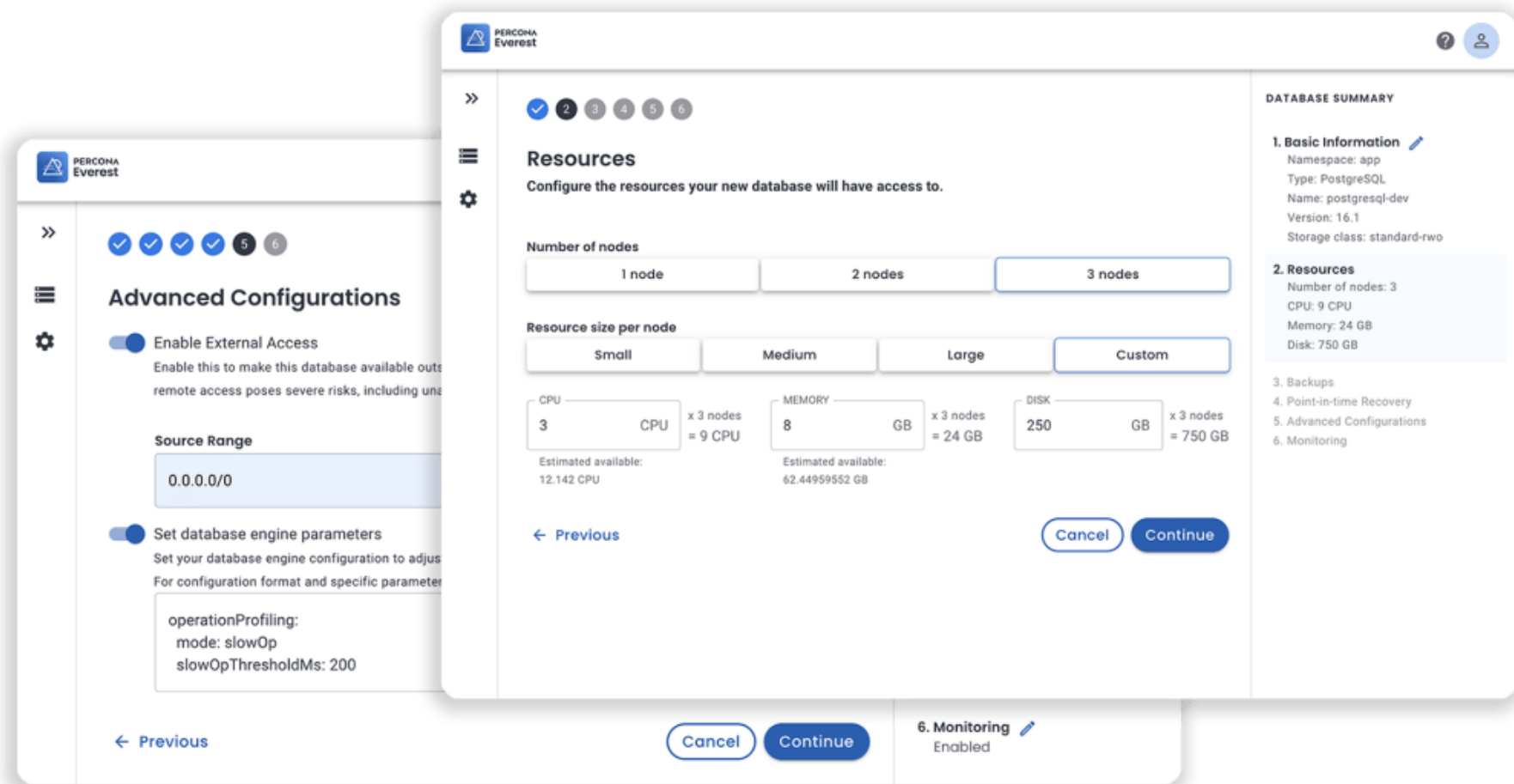
The screenshot displays the Stackgres management interface. On the left is a dark sidebar with navigation options: Namespace (gitlab-db), Stackgres Clusters, Instance Profiles, Configurations (Postgres, Connection Pooling, Managed Backups), Logs Servers, and Cluster Backups. The main content area shows the breadcrumb path: gitlab-db > SGClusters > gitlab-db > Status. Below this are links for 'SGCluster Documentation', 'Clone Cluster Configuration', 'Edit Cluster', and 'Delete Cluster'. A navigation bar includes 'Status' (selected), 'Configuration', 'Backups', 'Logs', and 'Monitoring'. The 'Cluster' section provides a summary: Total CPU (42000m, avg. load 2.39), Total Memory (150.00Gi), Primary Node Disk (246.08Gi / 500Gi), and Instances (3 / 3). The 'Pods' section contains a table with columns for Pod Name, Role, Status, CPU, Memory, Disk, and Containers.

Pod Name	Role	Status	CPU	Memory	Disk	Containers
gitlab-db-0	Primary	Active	14000m (avg. load 3.43)	50.00Gi	82.29Gi / 500.00Gi	6 / 6
gitlab-db-1	Replica	Active	14000m (avg. load 2.10)	50.00Gi	81.88Gi / 500.00Gi	6 / 6
gitlab-db-2	Replica	Active	14000m (avg. load 1.65)	50.00Gi	81.91Gi / 500.00Gi	6 / 6

Percona PostgreSQL Operator

- **Initially based off Crunchy Data Operator**
- **Apache 2.0 (using Open Source Images)**
- **Namespace and Cluster wide Modes**
- **Support Integration with PMM (Percona Monitoring and Management)**
- **Advanced Topologies Support**

Percona Everest as GUI



Best Practices



#1 Use
Operators

**For Production
Deployment you
need to ensure great
“Day 2” Automation**

#2 Setup
High
Availability

**Relying on Single
Instance in Kubernetes
environment is even
more dangerous**

#3 Keep
Persistent
Data
Persistent

Persistent Volumes; Local Disk or Fast Remote Storage

#4 Keep Data
per Pod Small

50TB of data
connected to single
POD is not a good
idea

#5 Use
Appropriate
Node Sizes

**Kubernetes or Not
Databases often
need “Big Iron” more
than Apps**

#6 Configure
Resource
Requests and
Limits

**Or you may have non
uniform Performance
and Severe Impact
on other Pods**

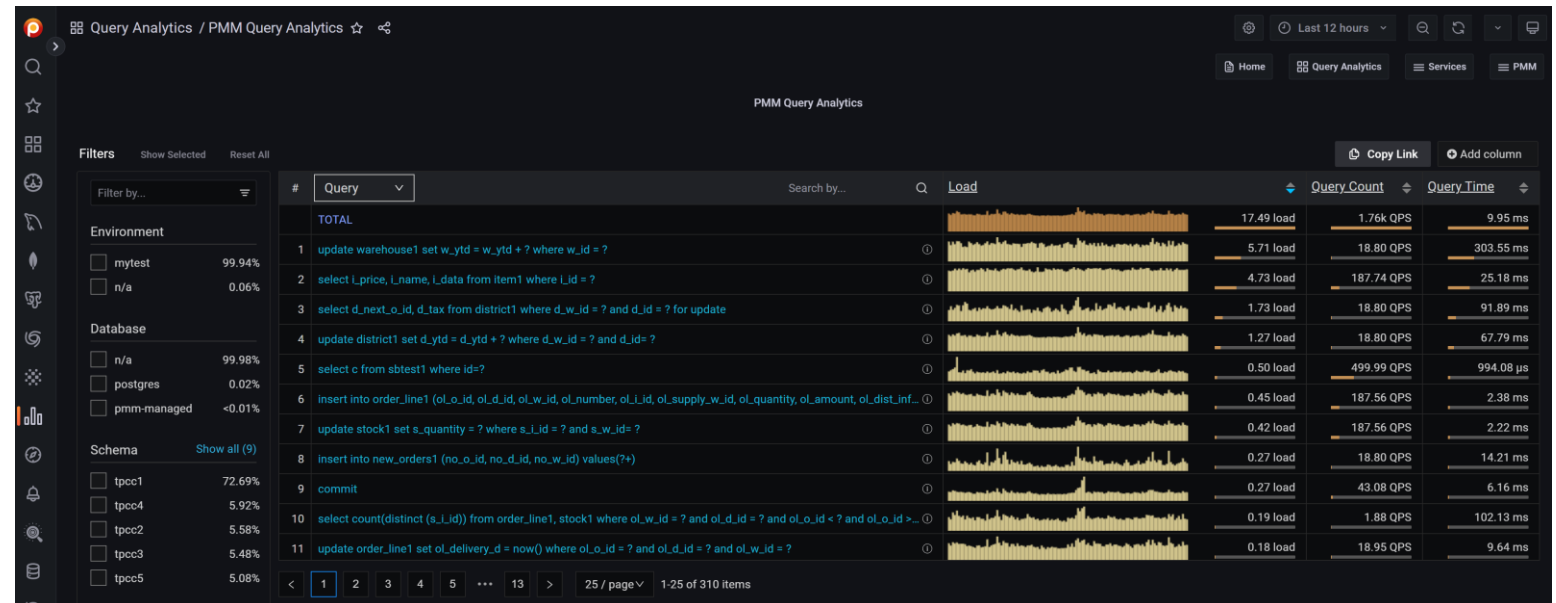
#7 Use Proper
Anti-Affinity

**3 Node Cluster Running
on Single Physical Node
is not Great High
Availability Solution**

#8 Tune your
Database

**Database
Configuration, Indexes,
Queries needs to be
taken care of as usually**

Query Analytics in Percona Monitoring and Management



#9
Understand
How to Scale

Some Databases can
be “Scaled Out” others
only “Scaled-Up” and
Scaled for Reads

#10 Control
Eviction with
Pod Priority

**Rescheduling Database
Pod Can be Expensive,
so better ensure it does
not happen too often**

#11 Do not
Expose your
Database
unless you
have to

**Unintended Publicly
Accessible Data is
leading cause of
Security Leaks**

#12 Enable
Encryption

**Data at Rest and
Data in Transit. Does
not cause huge
Overhead those days**

#13 Use
Kubernetes
Secrets

**Great way to pass
database access
credentials to your
application**

#14 Do not
forget
Backups

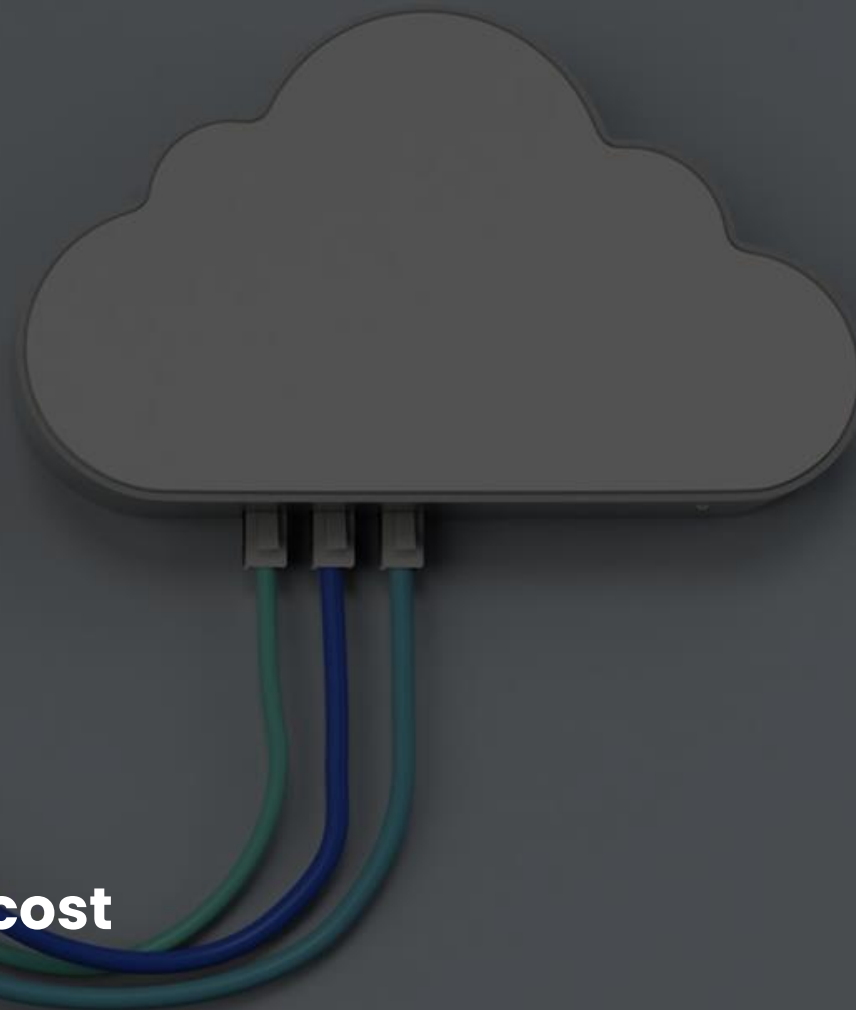
**Clustering Does not
Eliminate need for
backups. Do them. Good
Operators make it Easy**

#15 Consider
New
Generation
Databases

**Databases designed to be
run on Cloud Native
Infrastructure are Coming
- Neon, Oriole, Yugabyte**

#16 CPU Choices

Arm can be increasingly performant and cost effective in the cloud





#17 Pick Right Kubernetes deployment mode

**Managed and Self-Managed
both have their merits**



#18 Monitor Utilization

Spreading pods over more nodes than needed can be expensive

Thank you, Let's Connect!

<https://www.linkedin.com/in/peterzaitsev/>

<https://twitter.com/PeterZaitsev>

<http://www.peterzaitsev.com>