DIAMANTI

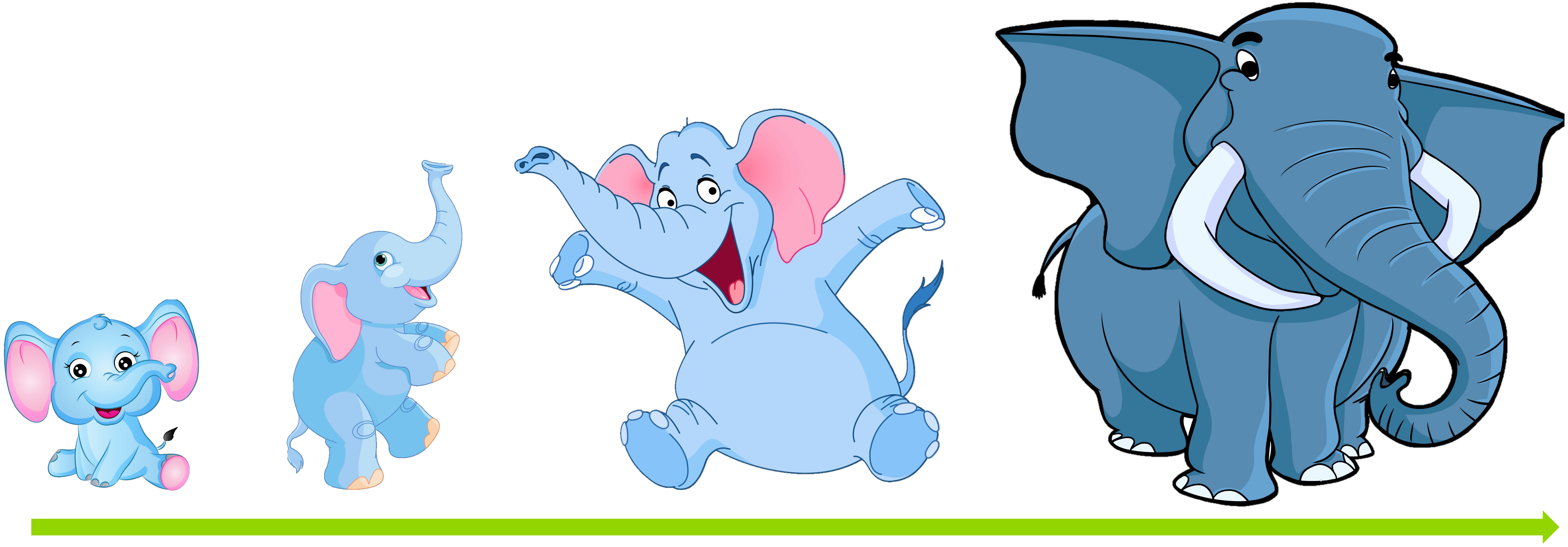
Unleashing the Power of PostgreSQL with Kubernetes and Diamanti

Arvind Gupta, Principal Solutions Architect, CKA certified

Agenda

- Tribute to PostgreSQL
- Cloud native landscape for Database-as-a-Service
- Containers, Kubernetes and Operators for Databases
- Importance of Hyperconverged platform for DBaaS
- PostgreSQL benchmarking results on Kubernetes and HCI

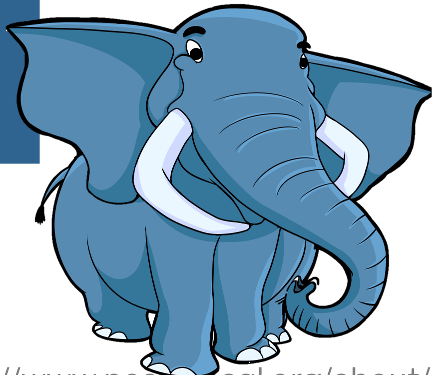
1986, University of California Berkeley



1986 2019

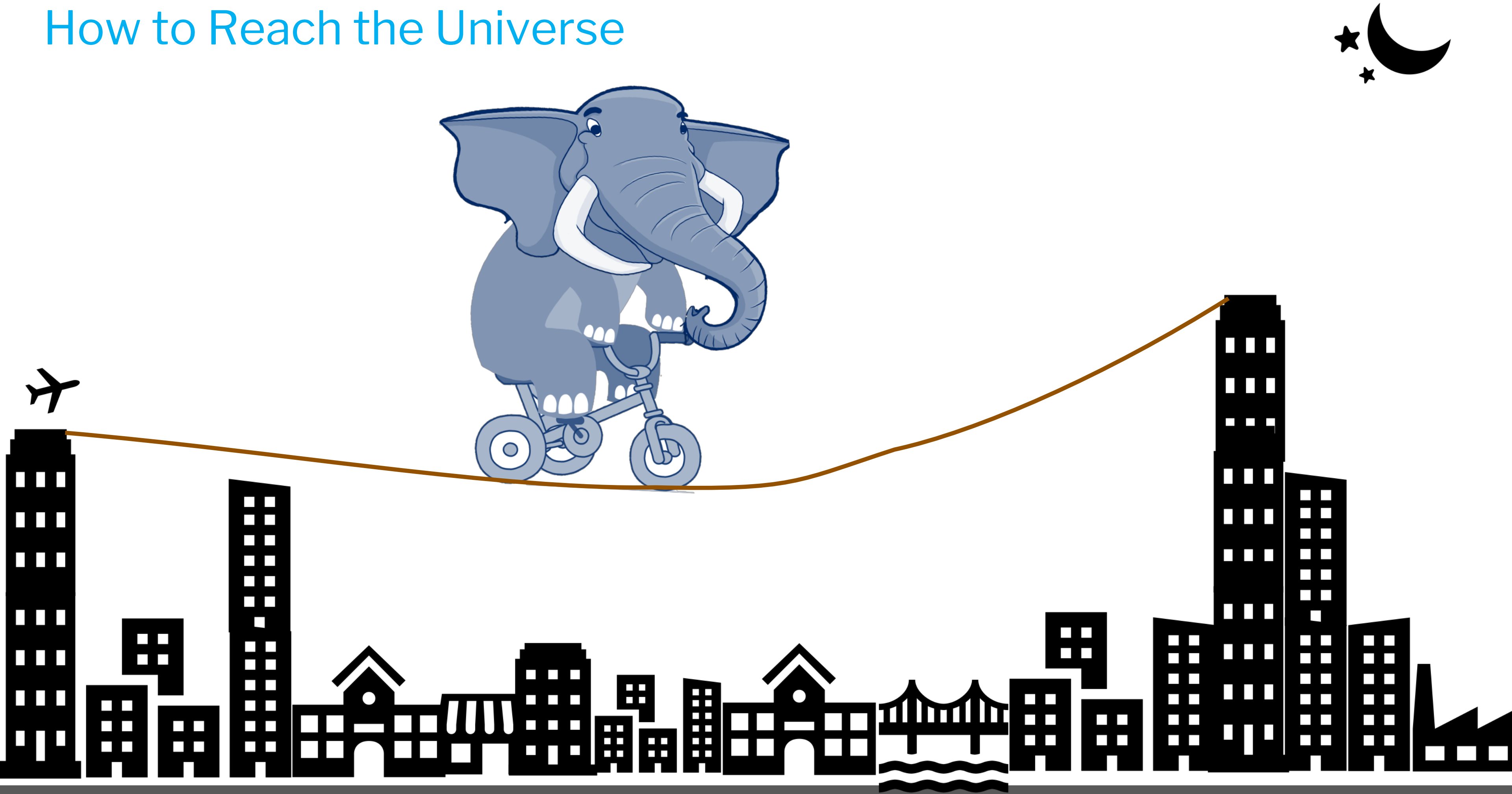
Some Key Stats About the Elephant

30+ Years Development	400+ Contributors	45,000+ Commits	50+ Local User Groups
1,100,000+ Lines of C	500+ Events	Millions of Happy Users	∞ Data Stored



Screenshot from https://www.postgresql.org/about/

How to Reach the Universe



Elephant Needs a Jetpack!

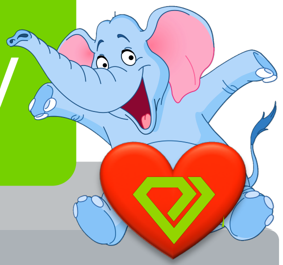
How Far Can the Elephant Go With a Jetpack?

Jetpack = Kubernetes + HCI

200K+ TPS*

900K+ IOPS

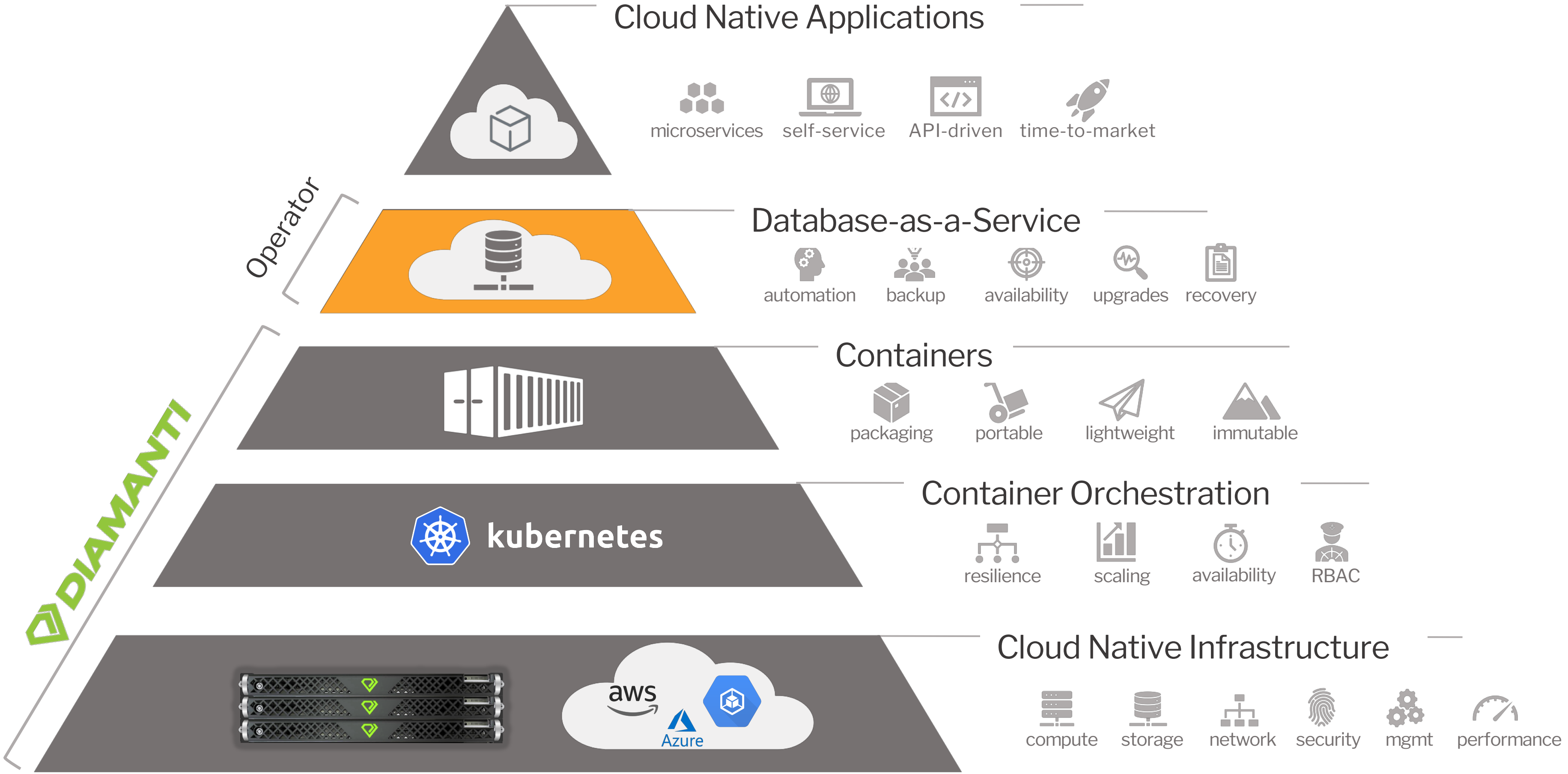
< 1ms Latency

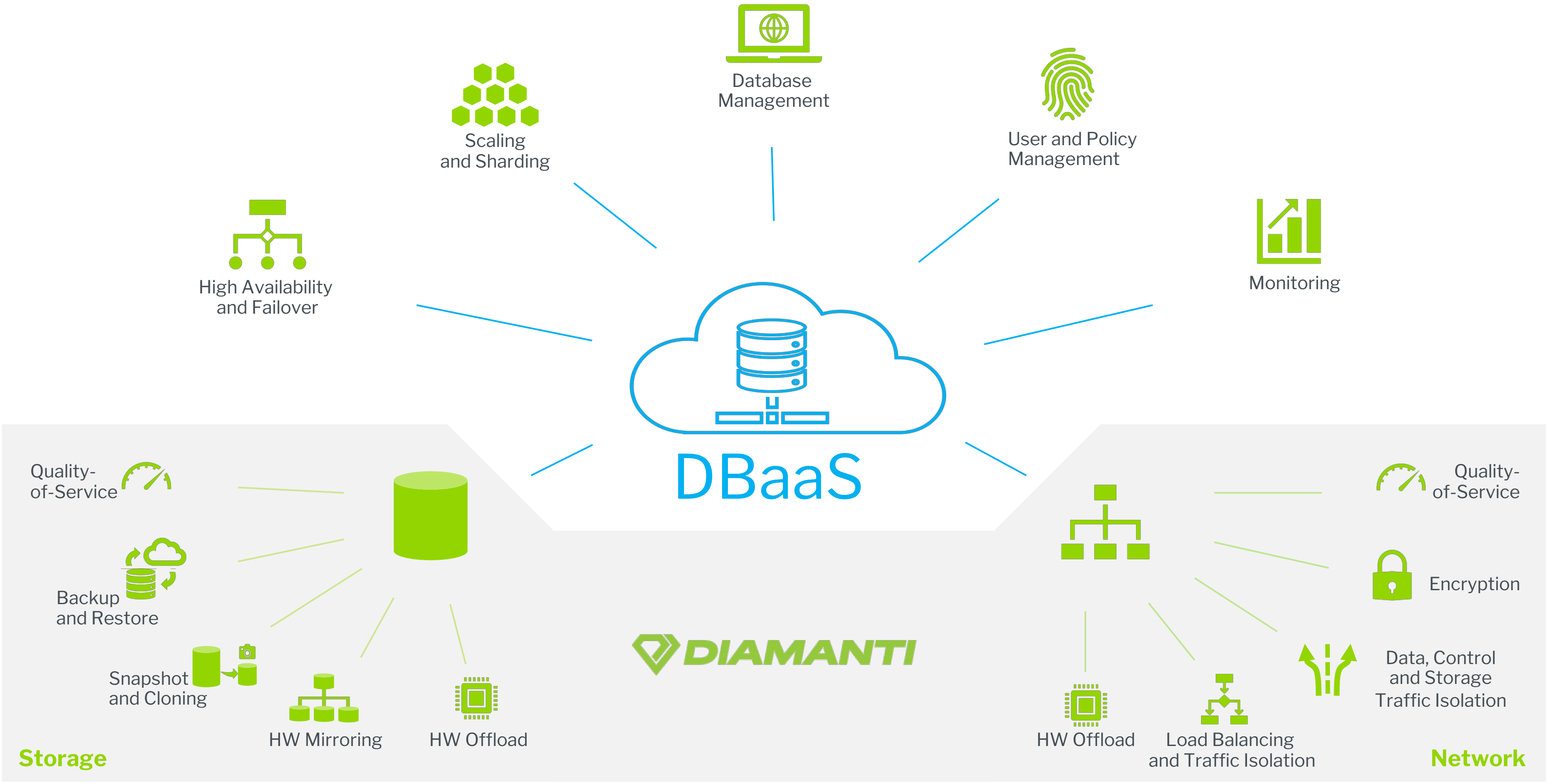


Per Diamanti HCI node

* Worst case scenario

The Building Blocks

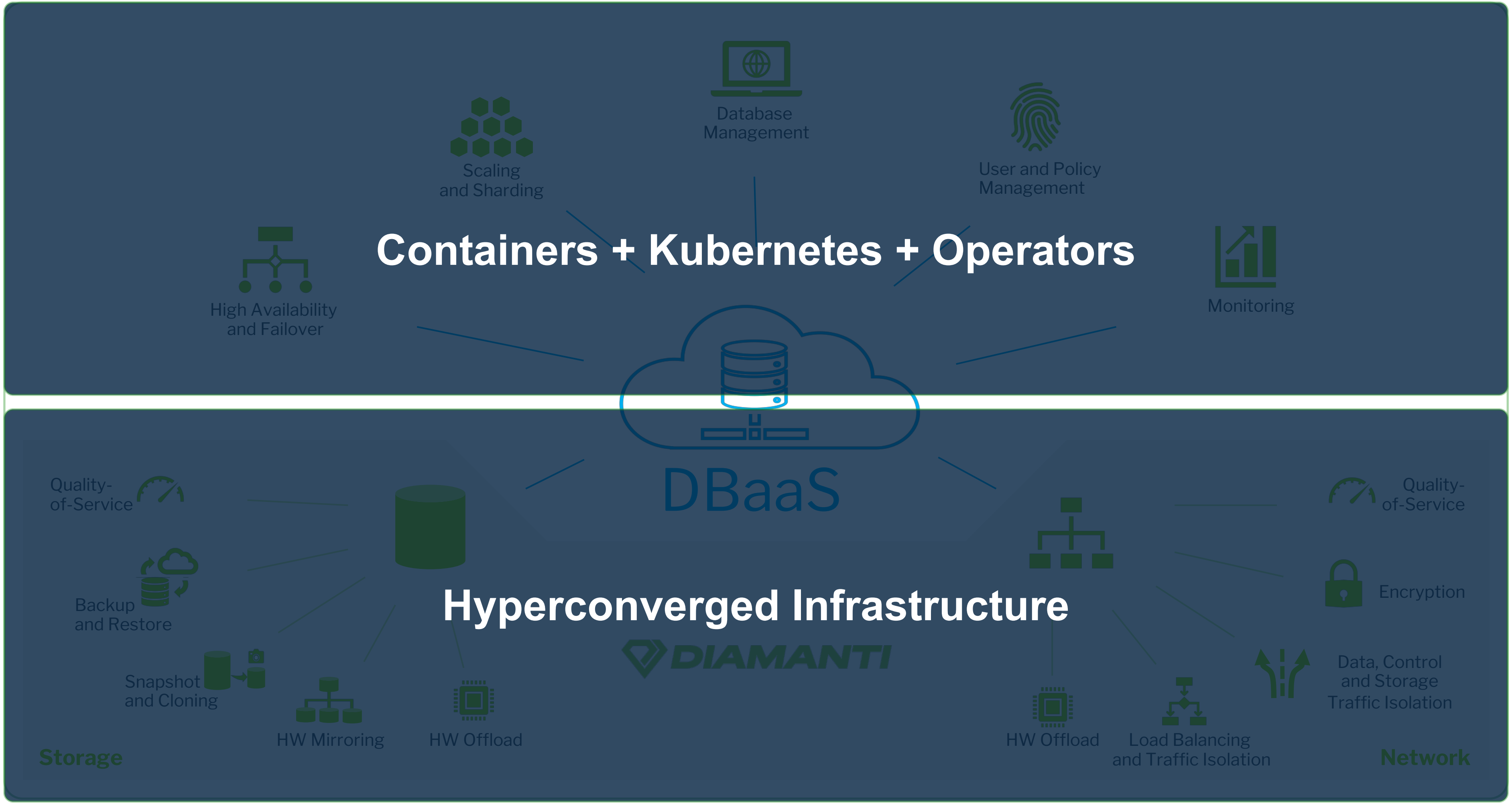




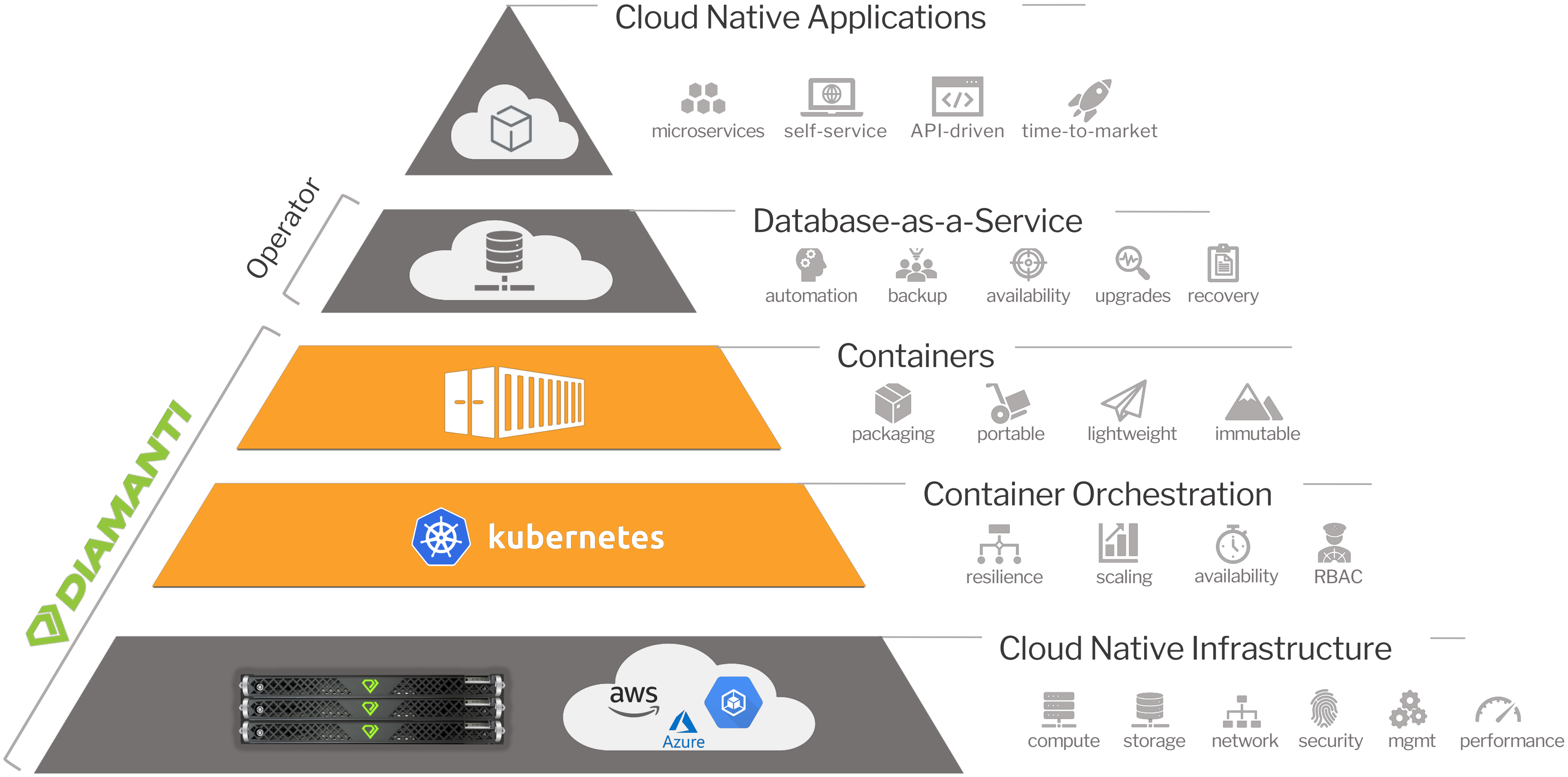
Database Management
Scaling and Sharding
User and Policy Management
High Availability and Failover
Monitoring
DBaaS
Quality-of-Service
Backup and Restore
Snapshot and Cloning
HW Mirroring
HW Offload
DIAMANTI
Quality-of-Service
Encryption
Data, Control and Storage Traffic Isolation
HW Offload
Load Balancing and Traffic Isolation
Storage
Network

Containers + Kubernetes + Operators

Hyperconverged Infrastructure

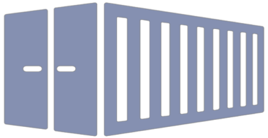


The Building Blocks



Containers and Kubernetes

- Container
  - Standard unit of application packaging
  - Independence from OS versions

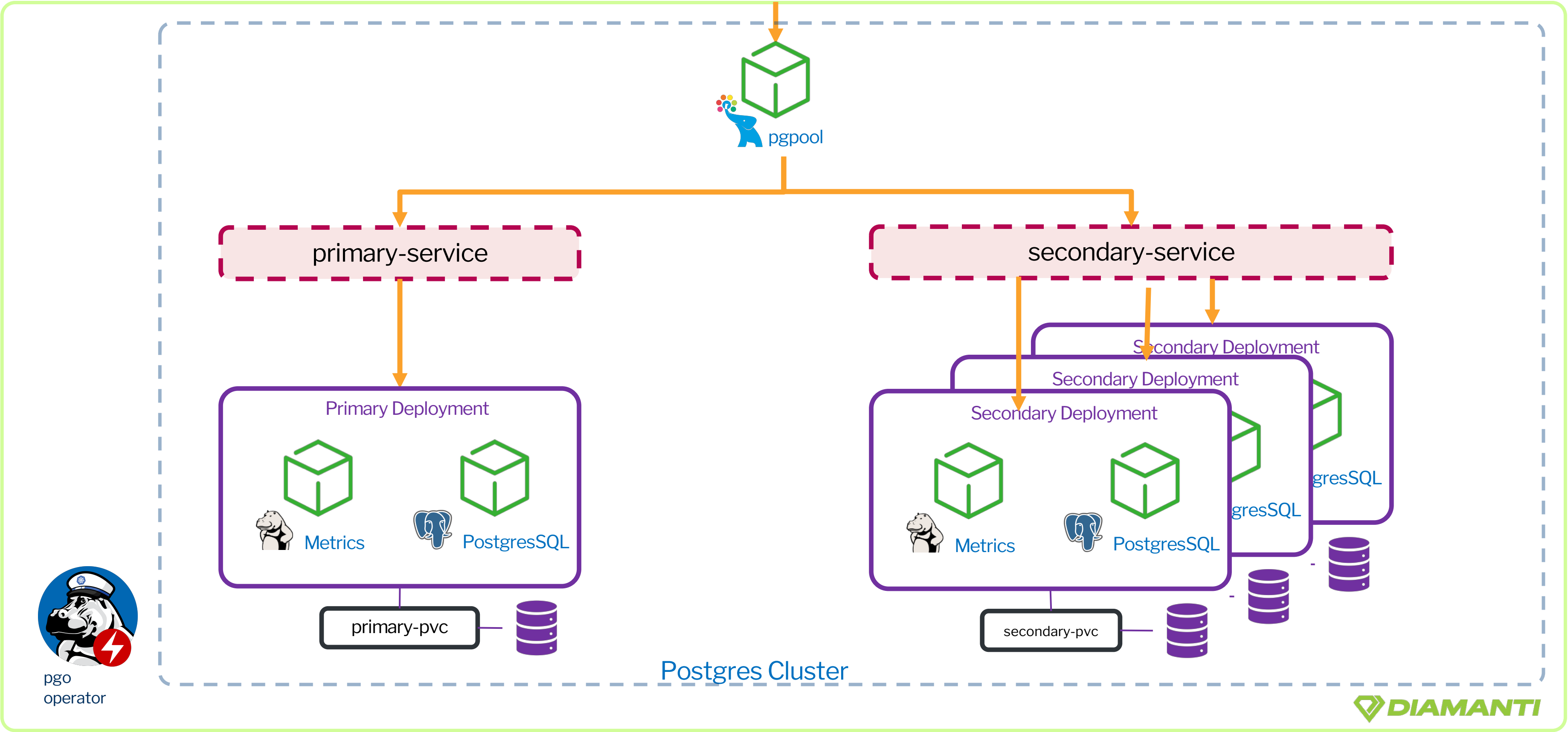


- Kubernetes
  - De facto container orchestration platform
  - Scalable and extensible

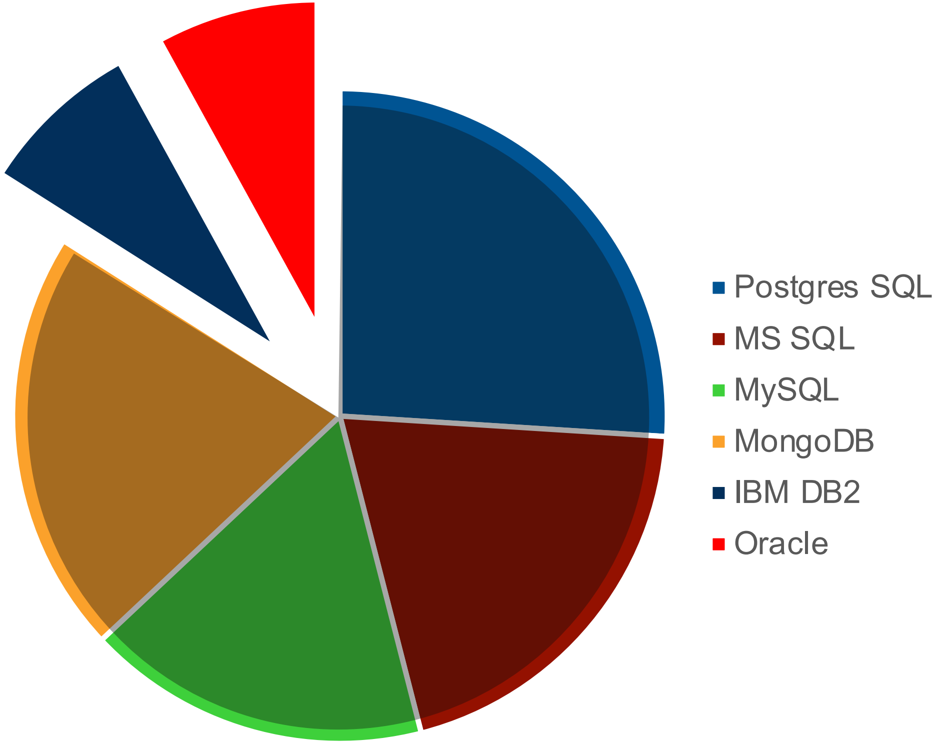
Why Kubernetes Operator?

- Operators is a purpose-built controller for life cycle management of specific Kubernetes application, with built-in operational knowledge
  - Automation: self-service, abstract complexities
  - Standardization: same workflow for different customization
  - Ease-of-use: user friendly CLI/API/UI for user interface
  - Flexibility: reuse and run your workload in any environment or cloud
- Database lifecycle management with an operator
  - Create, destroy or clone databases
  - Scale and sharding with automatic cloning and syncing
  - Setup high availability, replication, load balancing, failover
  - Setup backup, snapshots, disaster recovery
  - User and policy management
  - Monitoring

Highly Available Postgres Cluster Deployed with Operator



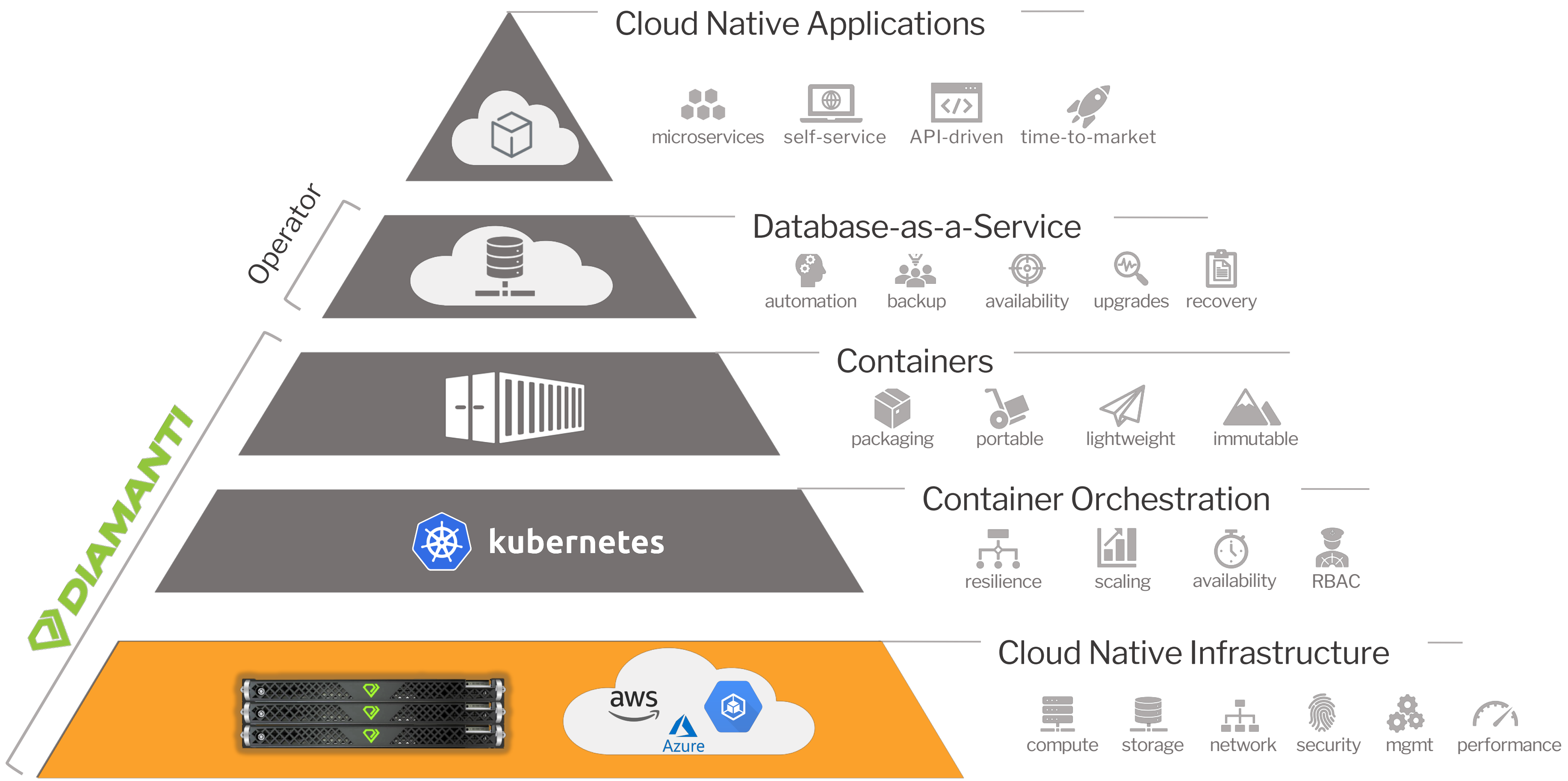
Typical Mix of Databases in an Enterprise



85%

Databases have Kubernetes Operators

The Building Blocks

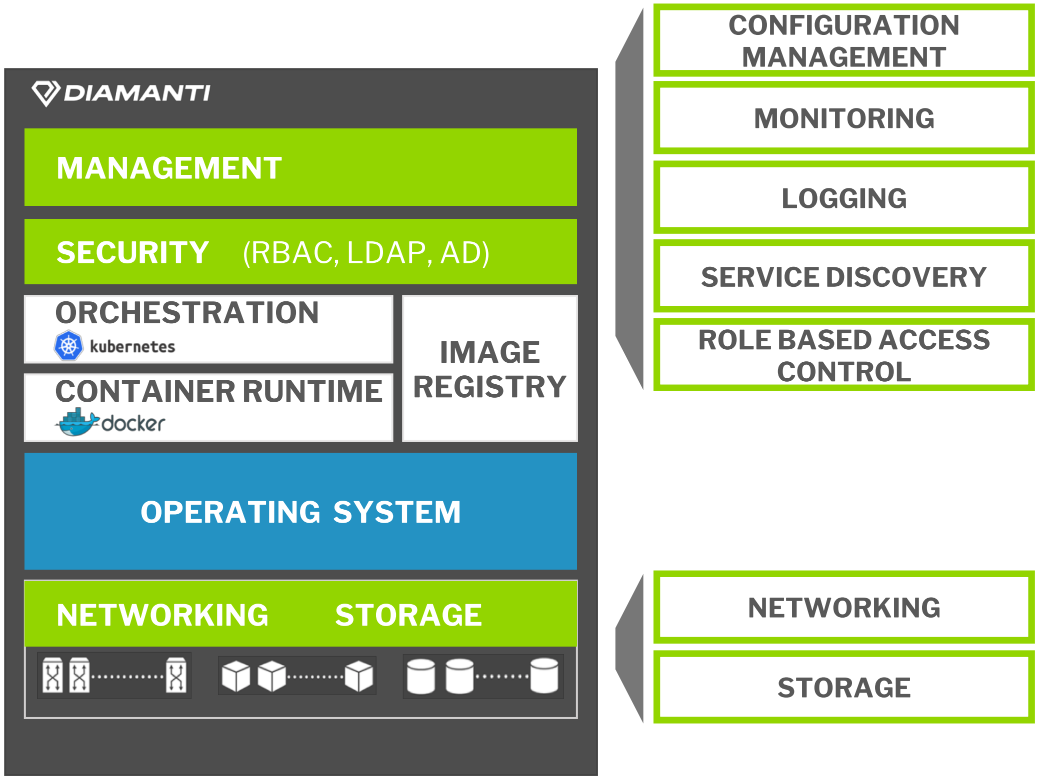


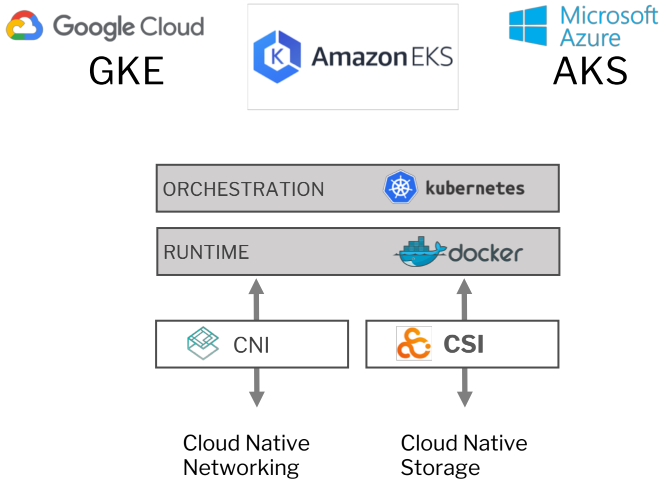
Infrastructure Sprawl – OS, Servers, Storage

>12	+	>10	+	>4
Different OS versions		Different server configurations		Teams handling different lifecycles

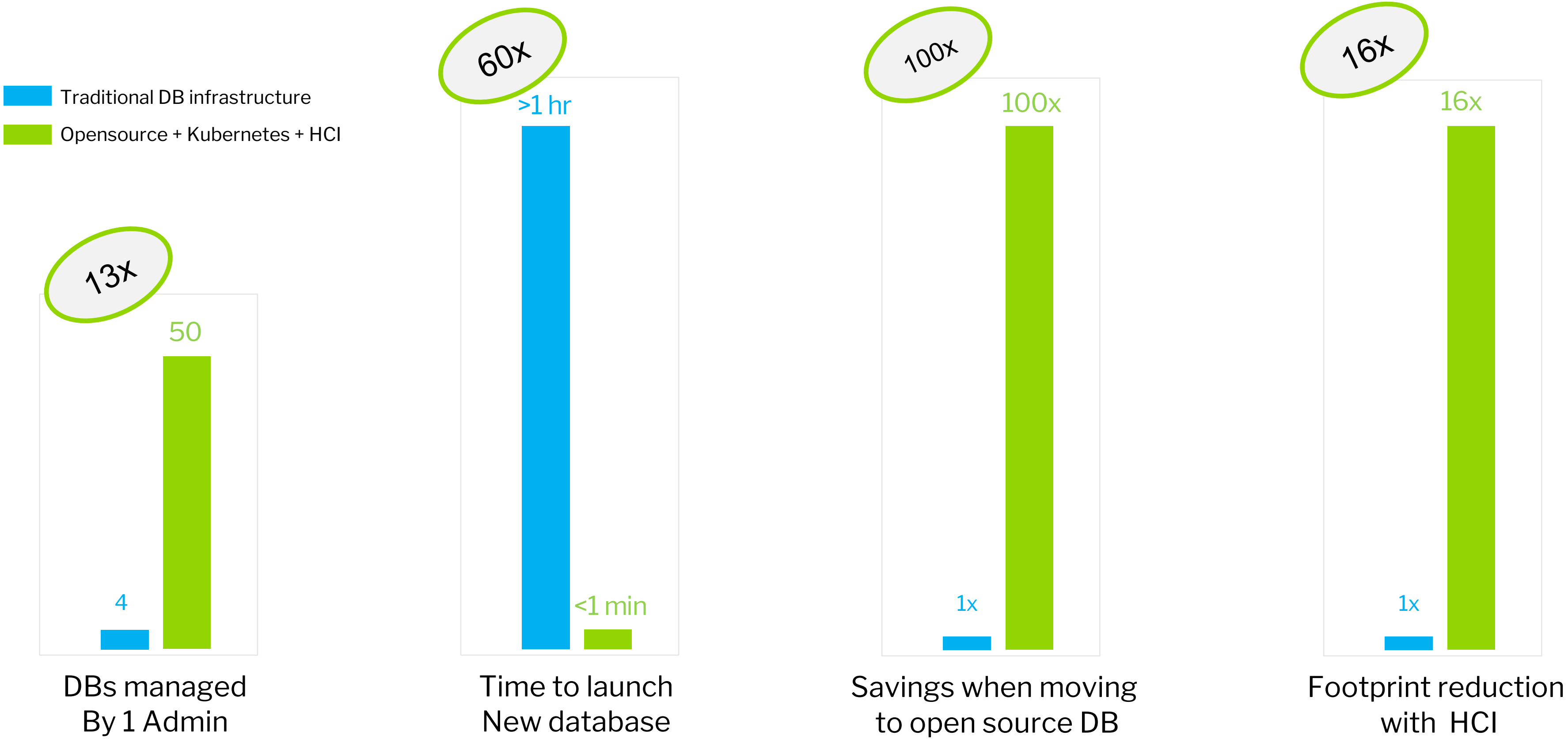
= Super high OpEx cost

Diamanti Enterprise Kubernetes Platform





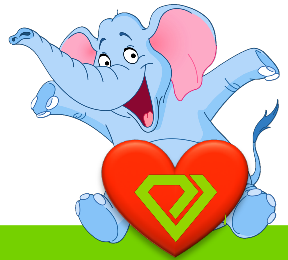
Significant Gains with Open Source DBs on Kubernetes and HCI



DIAMANTI

Performance

Benchmarking: Select Only Queries



200K+ TPS*

900K+ IOPS

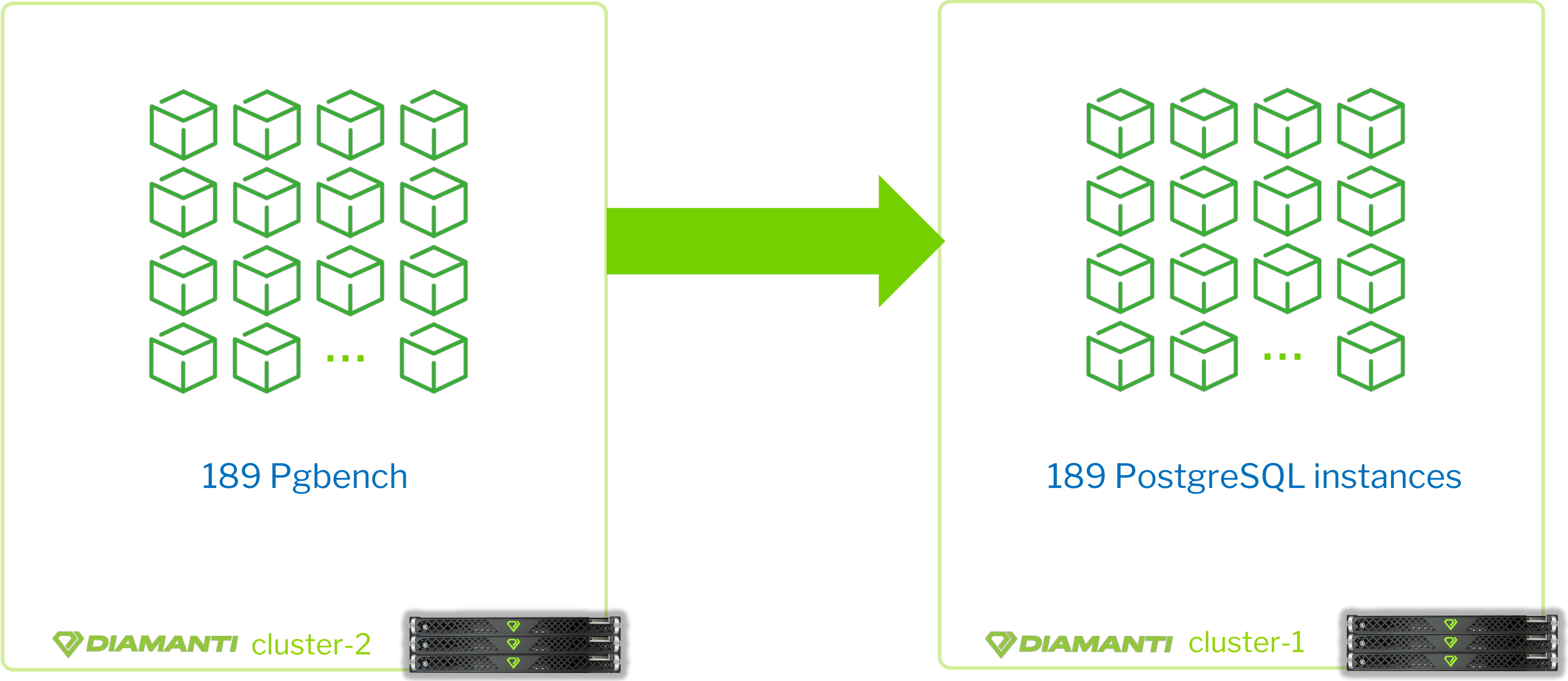
< 1ms Latency

DIAMANTI

Per Diamanti HCI node

* Worst case scenario

Postgres Benchmark Setup



Postgres Benchmark Setup: Hardware Configuration

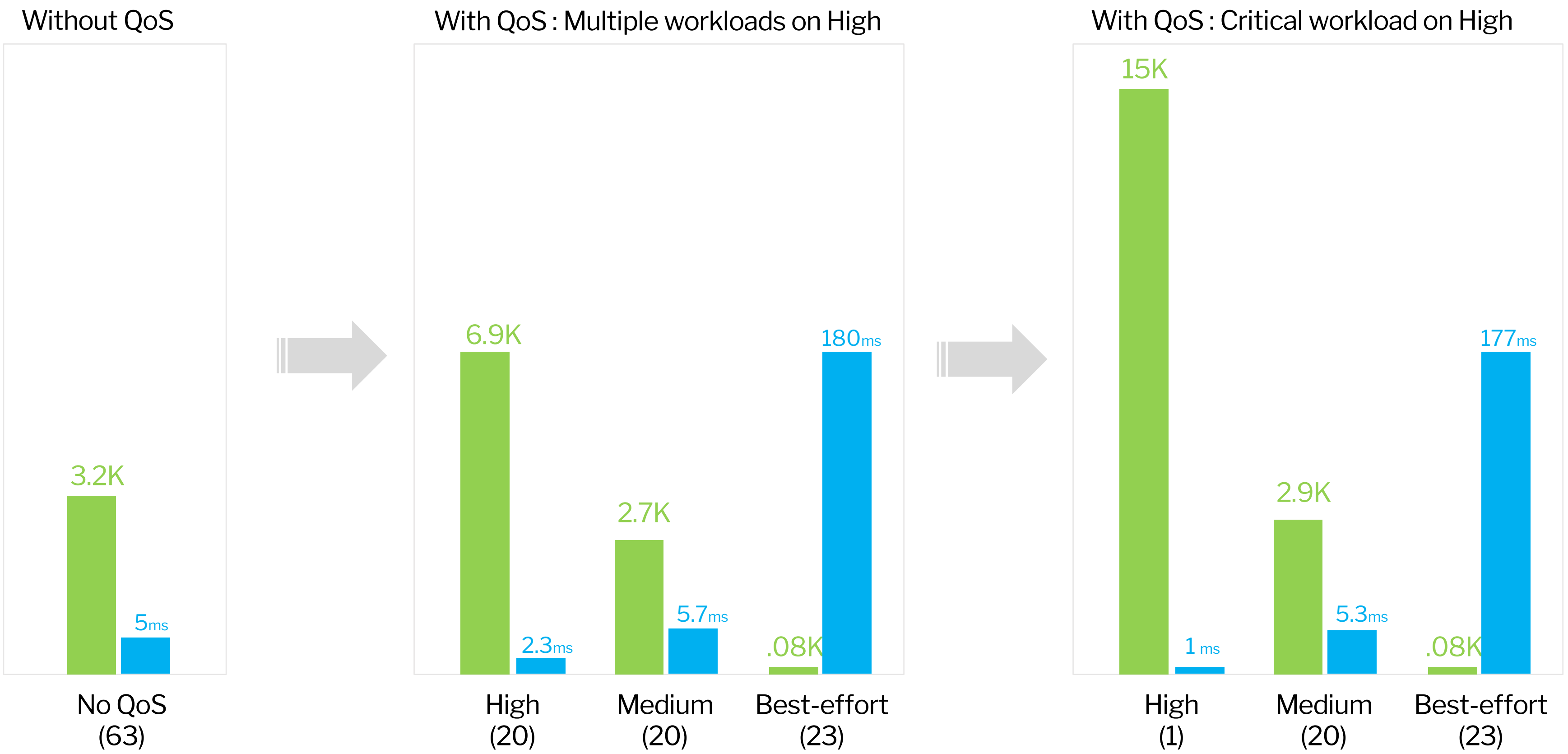
Cluster configuration:
Number of nodes: 3
Total PostgreSQL pods: 189
QoS: 945k provisioned IOPS

Node configuration:
CPUs: 64 HT Cores
Memory: 256 GB of RAM
Storage: 3TB
VNICs: 63
Total PostgreSQL pods: 63
QoS: 315k provisioned IOPS

PostgreSQL pod configuration:
Image: single-instance PostgreSQL
CPU: 1HT
RAM: 3.9GB
Volume size: 20GB
Data load factor: 600
Capacity used: 10-12 GB

QOS configuration:
High: 20k provisioned IOPS per pod
Medium: 5k provisioned IOPS per pod
Best-efforts: NO provisioned IOPS per pod

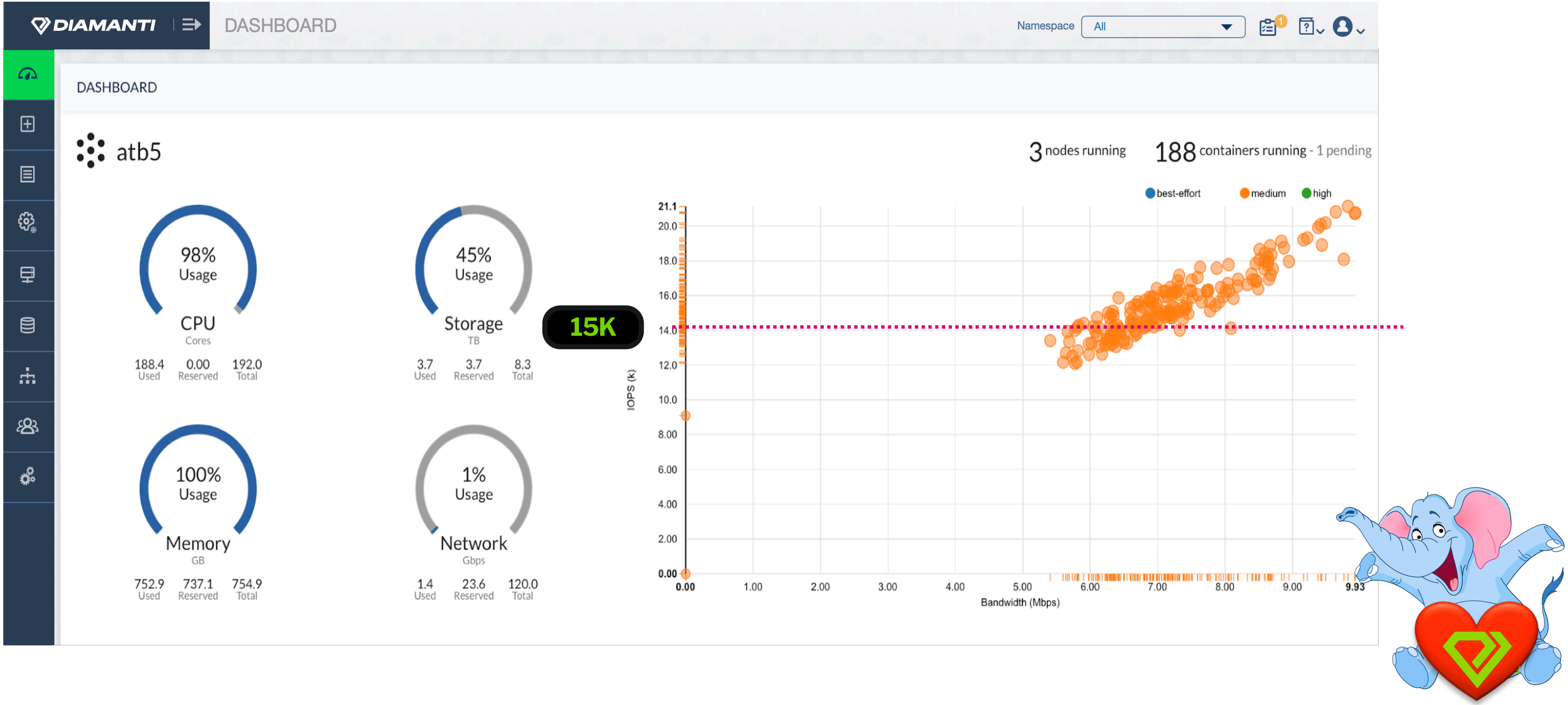
TPS Comparisons: Meet your SLAs, GUARANTEED!



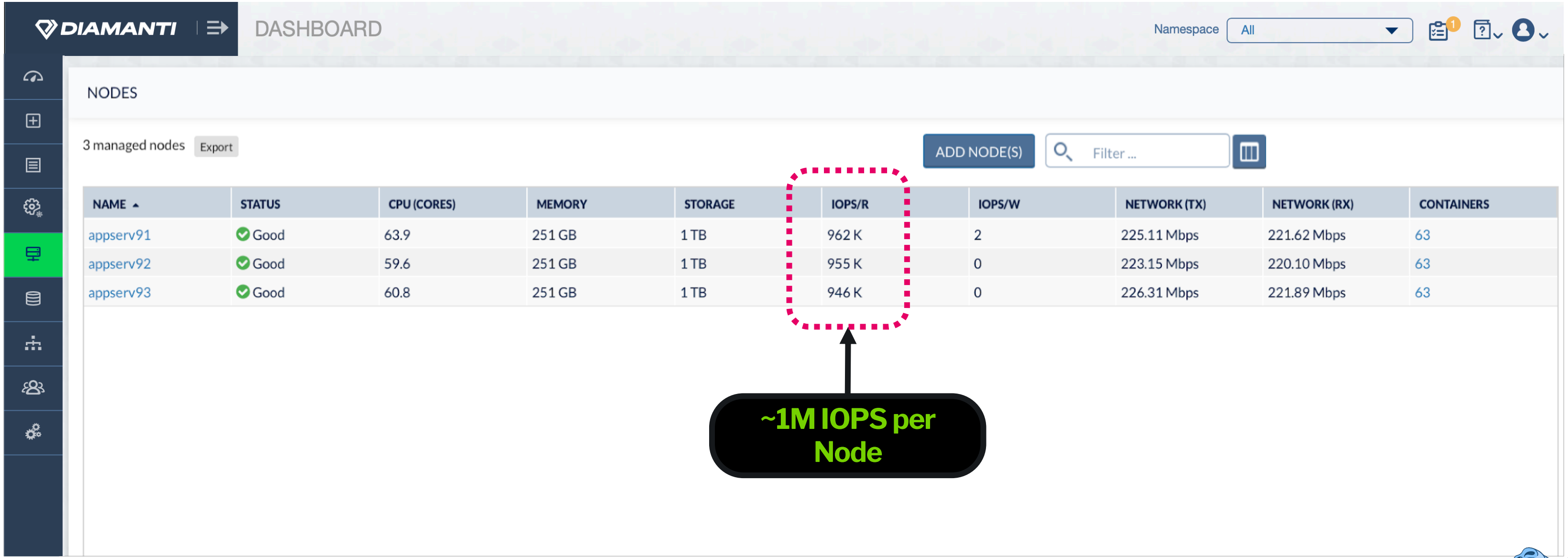
TPS Comparisons: Meet your SLAs, GUARANTEED!

Query type		Without QOS	With QOS (multiple high)			With QOS (critical load on high)		
		(63 per node)	High (20 per node)	Medium (20 per node)	Best-effort (23 per node)	High (1 per node)	Medium (20 per node)	best-effort (23 per node)
Select Only	PS	3.2k	6.9k	2.7k	88	15k	2.9k	89
	Latency	5 ms	2.3	5.7	180 ms	1 ms	5.3 ms	177 ms
TPC-B	PS	600	1.5k	1.3k	14	7.4k	1.5k	15
	Latency	26 ms	10 ms	12 ms	1119 ms	2.1 ms	11 ms	1108 ms

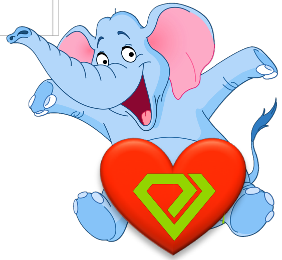
Performance Results - Select Only Queries



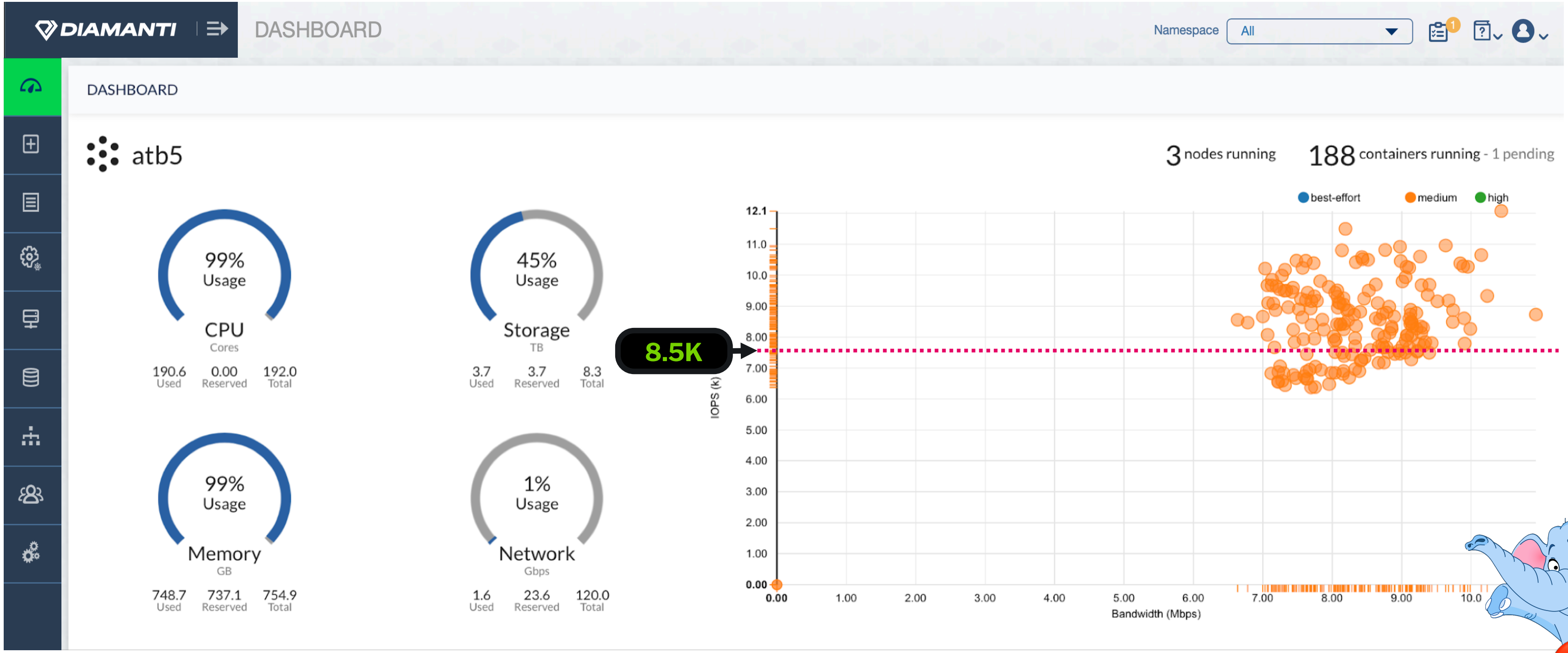
Performance Results - Select Only Queries

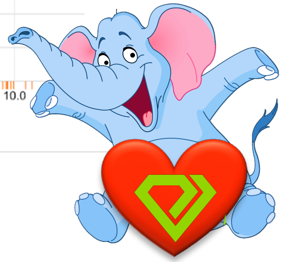


NAME	STATUS	CPU (CORES)	MEMORY	STORAGE	IOPS/R	IOPS/W	NETWORK (TX)	NETWORK (RX)	CONTAINERS
appserv91	Good	63.9	251 GB	1 TB	962 K	2	225.11 Mbps	221.62 Mbps	63
appserv92	Good	59.6	251 GB	1 TB	955 K	0	223.15 Mbps	220.10 Mbps	63
appserv93	Good	60.8	251 GB	1 TB	946 K	0	226.31 Mbps	221.89 Mbps	63

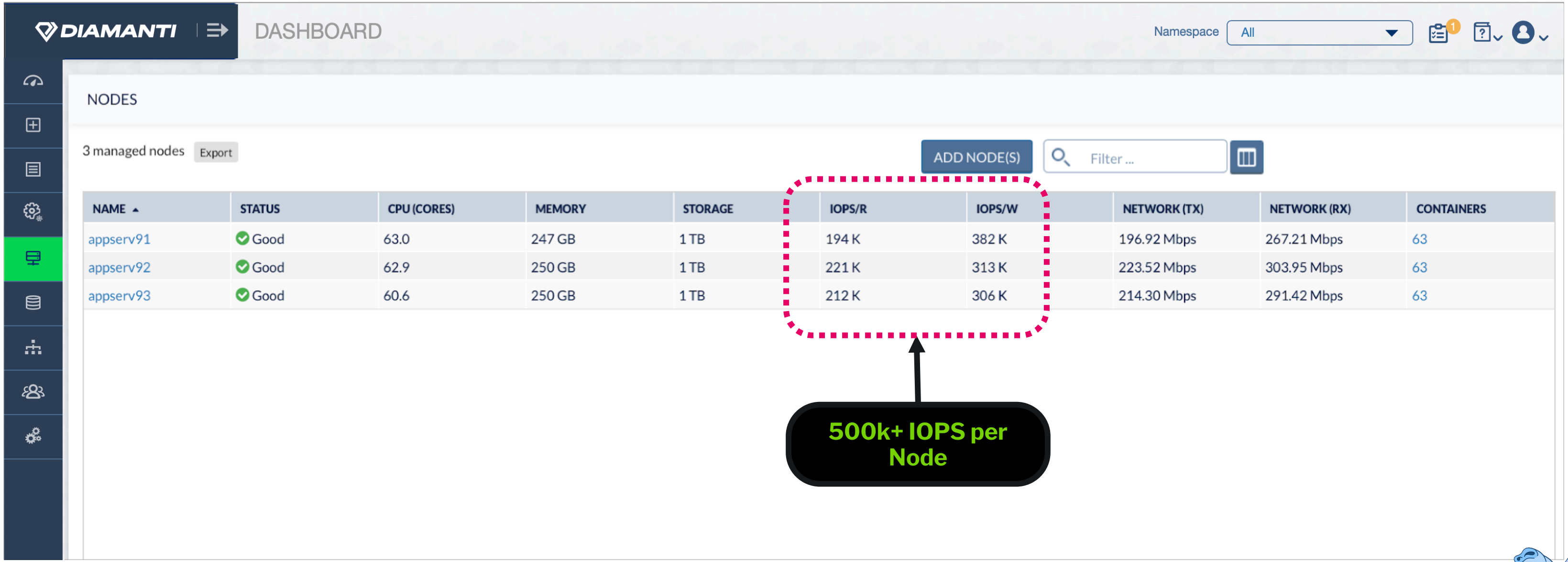


Performance Results - TPC-B Like Queries

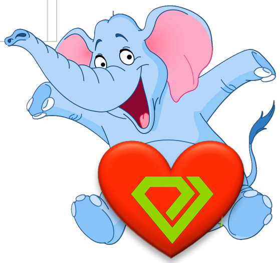




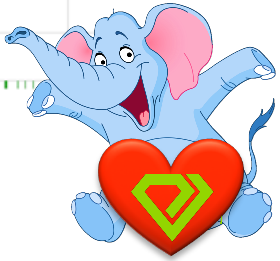
Performance Results - TPC-B Like Queries



NAME	STATUS	CPU (CORES)	MEMORY	STORAGE	IOPS/R	IOPS/W	NETWORK (TX)	NETWORK (RX)	CONTAINERS
appserv91	Good	63.0	247 GB	1 TB	194 K	382 K	196.92 Mbps	267.21 Mbps	63
appserv92	Good	62.9	250 GB	1 TB	221 K	313 K	223.52 Mbps	303.95 Mbps	63
appserv93	Good	60.6	250 GB	1 TB	212 K	306 K	214.30 Mbps	291.42 Mbps	63

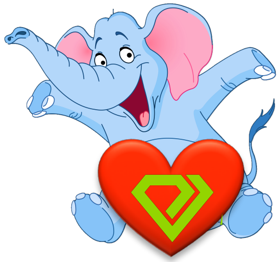


The QoS Impact - Select Only Queries

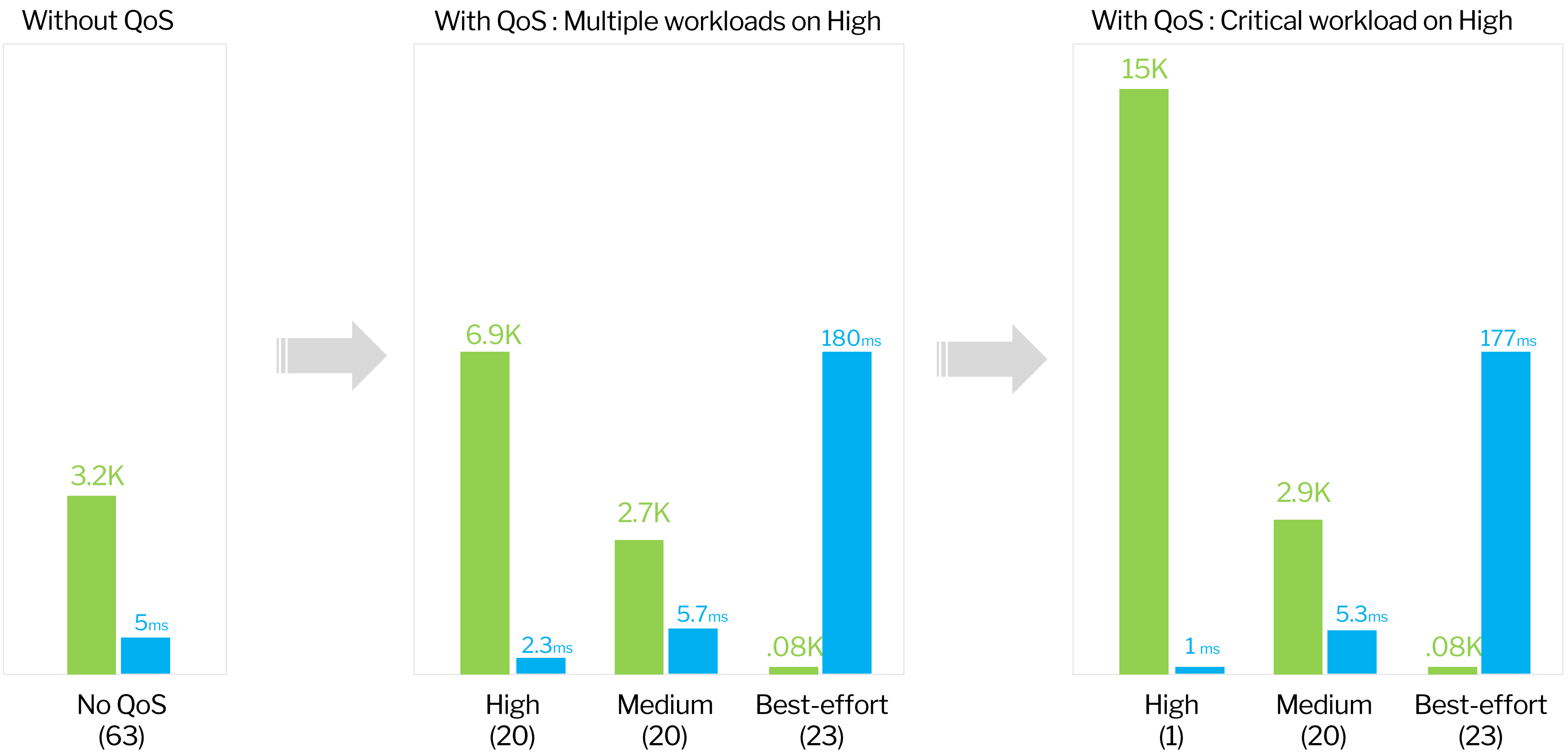


The QoS Impact - TPC-B Like Queries

The QoS impact: Guard you critical pods



TPS Comparisons: Meet your SLAs, GUARANTEED!



Learn More at

www.diamanti.com

info@diamanti.com

@diamanticom

https://www.linkedin.com/company/diamanti

http://bit.ly/DiamantiDBaaS

DIAMANTI

Thank You!

DIAMANTI

Supporting slides

Postgres bench results

Singe instance PostgresSQL with no CPU limit : Read only

```
> pgbench -r -S --host=172.16.225.36 --port=5432 --username=primaryuser --jobs=2 --time=180 --
client=32 pgbench
running iteration 1 on 172.16.225.36
starting vacuum...end.
transaction type: <builtin: select only>
scaling factor: 600
query mode: simple
number of clients: 32
number of threads: 2
duration: 180 s
number of transactions actually processed: 16935308
latency average = 0.340 ms
tps = 94080.942801 (including connections establishing)
tps = 94082.575020 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set aid random(1, 100000 * :scale)
         0.340  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Singe instance PostgresSQL with 4G mem : Read only

```
pgbench -r -S --host=172.16.225.18 --port=5432 --username=primaryuser --jobs=2 --time=300 --client=32 pgbench
running iteration 1 on 172.16.225.18
starting vacuum...end.
transaction type: <builtin: select only>
scaling factor: 600
query mode: simple
number of clients: 32
number of threads: 2
duration: 300 s
number of transactions actually processed: 15504631
latency average = 0.619 ms
tps = 51681.884356 (including connections establishing)
tps = 51682.515420 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set aid random(1, 100000 * :scale)
         0.619  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Singe instance PostgresSQL with 4G mem : TPC-B like queries

```
pgbench -r --host=172.16.225.18 --port=5432 --username=primaryuser --jobs=2 --time=300 --client=32 pgbench
running iteration 1 on 172.16.225.18
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 600
query mode: simple
number of clients: 32
number of threads: 2
duration: 300 s
number of transactions actually processed: 2621329
latency average = 3.662 ms
tps = 8737.683240 (including connections establishing)
tps = 8737.779495 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set aid random(1, 100000 * :scale)
         0.000  \set bid random(1, 1 * :scale)
         0.000  \set tid random(1, 10 * :scale)
         0.000  \set delta random(-5000, 5000)
         0.174  BEGIN;
         1.126  UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
         0.268  SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
         0.280  UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
         0.300  UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
         0.235  INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid,
:delta, CURRENT_TIMESTAMP);
         1.278  END;
```

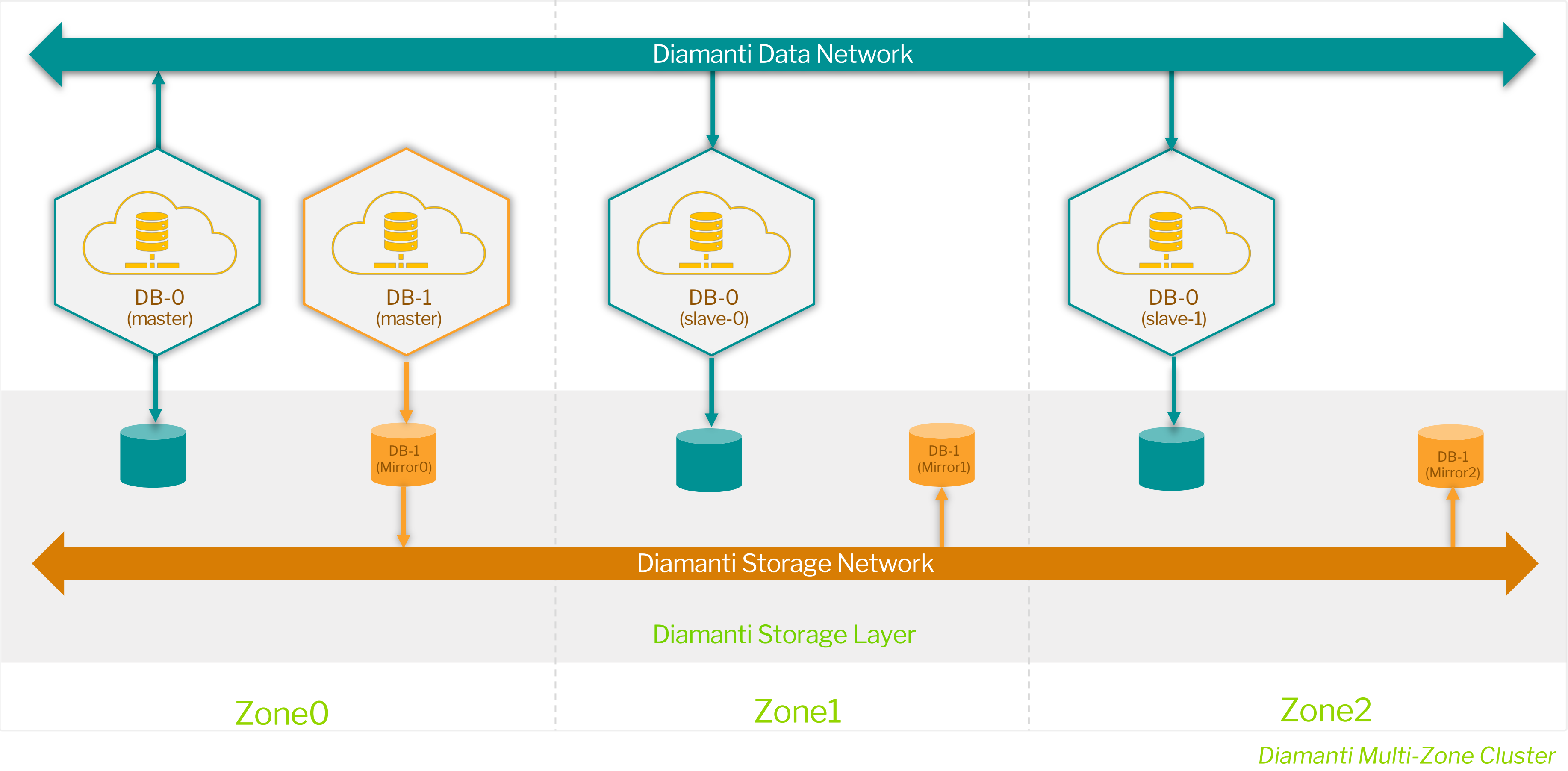
DIAMANTI

Why Diamanti

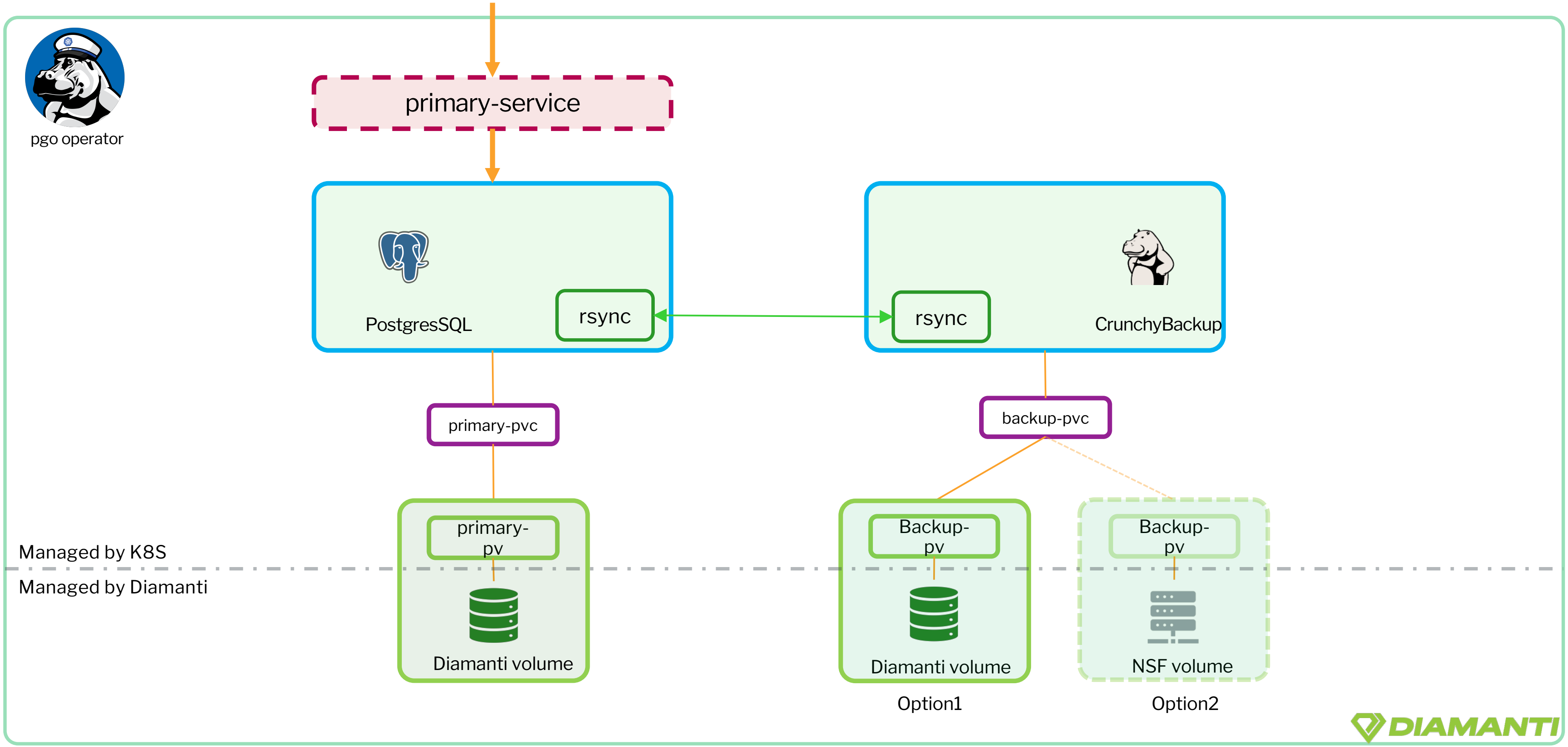
Why run DBaaS on Diamanti?

- Ready to go Kubernetes platform in under 15 minutes
- QoS:
  - No noisy neighbors
  - Isolation at the PCI-e layer
  - Guaranteed performance
  - Integrated with k8s storage class
- Performance:
  - Fast NVMe storage with million+ IOPS
  - 300 us latency across the cluster (at full load)
- Mirroring/ HW level replication:
  - Diamanti Mirroring (across D20 nodes)
  - Diamanti Backup Controller for snapshot based backups
  - Separate storage network (for NVMeoE traffic)
  - Quick recovery of pods in case of failure.
- Simplified L2 network
  - Direct TCP connectivity to pods via data network.
  - No need to route traffic via host/nodes
  - Separate control and data plane
  - Easy communication between replicas.
- Multi-zone(AZs) support for better HA and DR
  - Diamanti network supports stretched/campus clusters; and schedules pods with storage across multiple zones for HA.
  - Scheduler take care of volume locality.

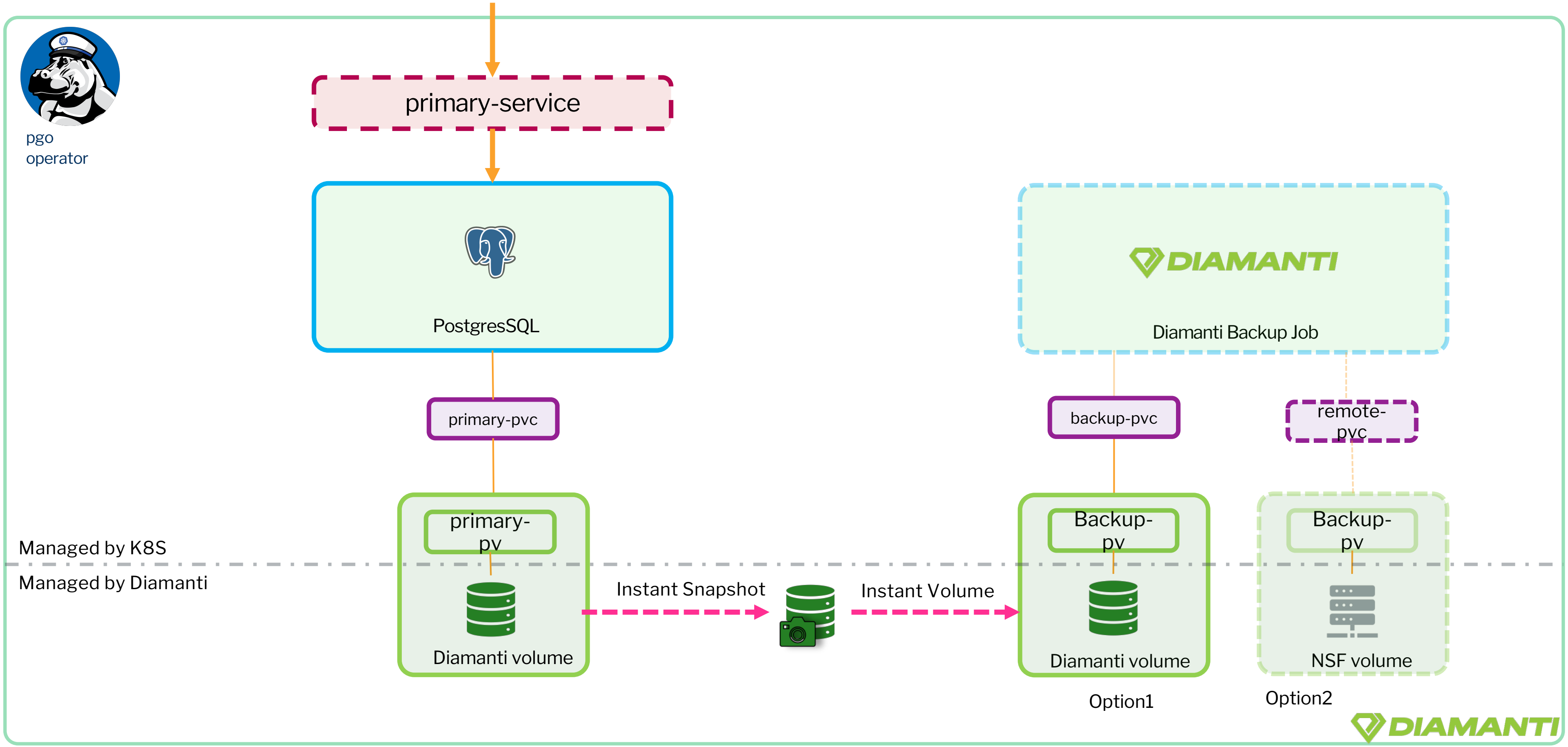
High Availability across Diamanti Multi-Zone clusters



Postgres Operator by Crunchy Data: Crunchy Backup



Postgres Operator by Crunchy Data: Diamanti Backup



DIAMANTI

Thank You!