“

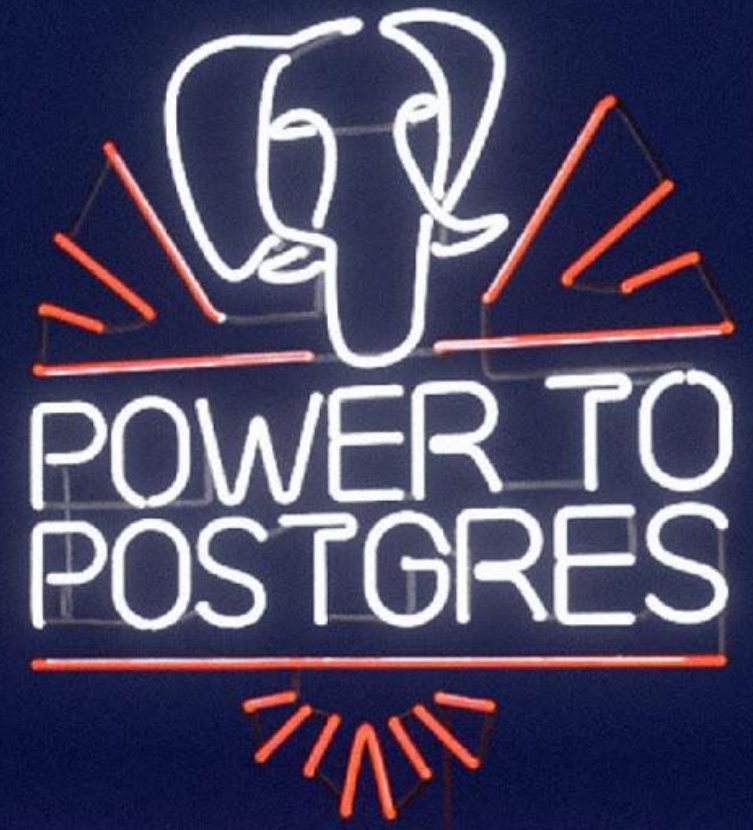
Perché dovrei avere un database su Kubernetes?

“

Perché dovrei usare Postgres come database?

Perché dovrei fare un benchmark del database prima di andare in produzione?

Benchmarking Cloud Native PostgreSQL



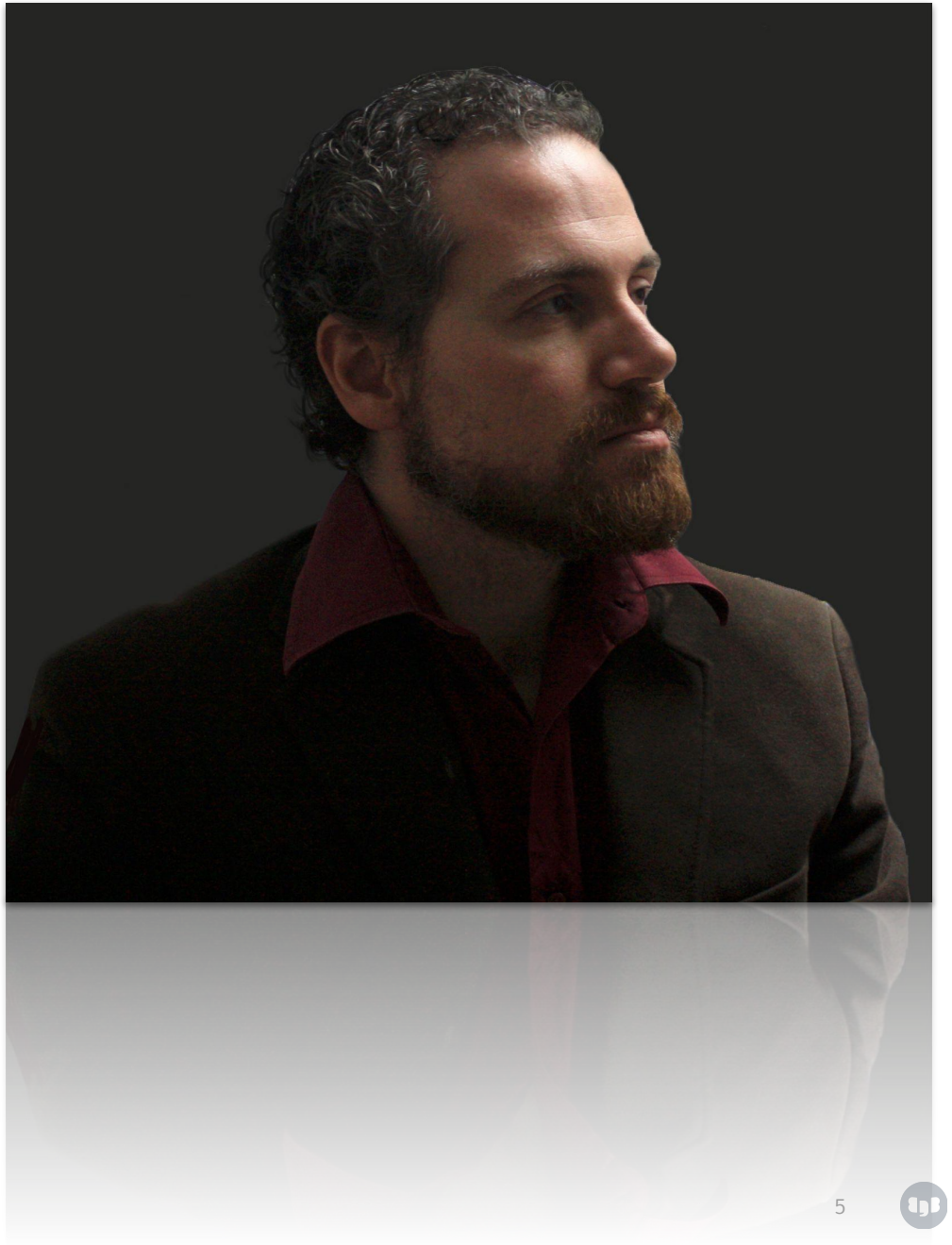
Francesco Canovai

1 Luglio 2021

EDB

Francesco Canovai

- PostgreSQL
- Infrastruttura
- Automazione
- Testing
- Kubernetes



Twitter: @fcanovai / @EDBPostgres

Argomenti

- I “perché”
- Com’è iniziato
- Storage
- Database
- cnp-bench
- Risultati
- Conclusioni

Intro
I “Perché”

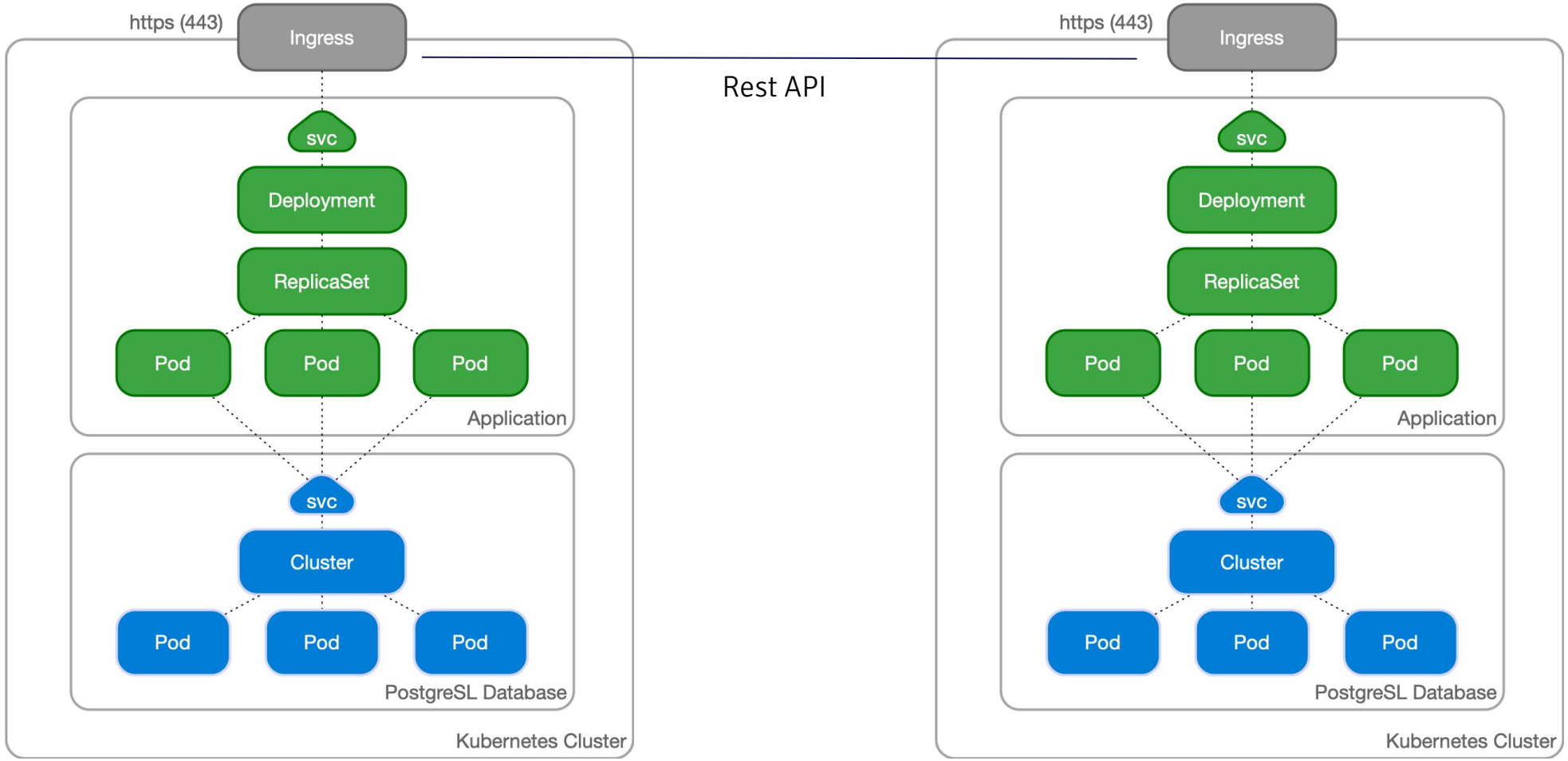
Perché un database su Kubernetes

Utilizzare al meglio un'infrastruttura a microservizi

- Un microservizio dovrebbe essere il proprietario dei suoi dati
  - I dati devono essere organizzati in un singolo data store o database
  - Solo il servizio deve avere accesso ai dati
  - Gestione dello schema inclusa
- Riduce il **Carico cognitivo del Team**
  - "Software that fits in your head" (Dan North)
- Definire **chiare interfacce di comunicazione** tra microservizi
- In Kubernetes, le applicazioni e i database coesistono:
  - **Tutto è un workload** (incluso il database)
  - Container e microservizi sono parte della definizione di Cloud Native

Esempio di microservizi

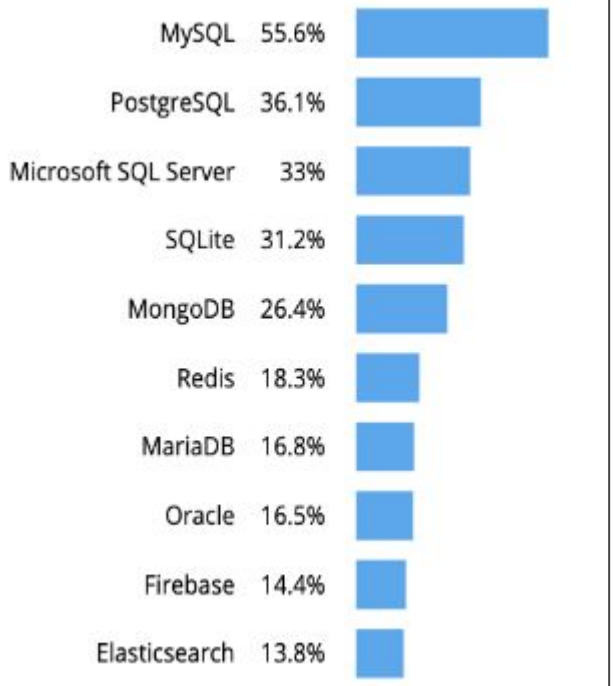
Applicazioni web con backend PostgreSQL



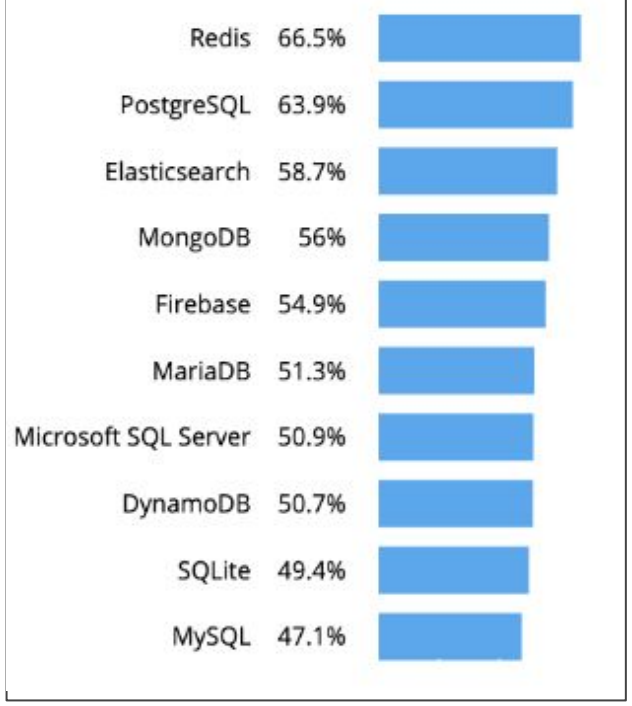
Perché PostgreSQL?

Chi lo prova lo ama

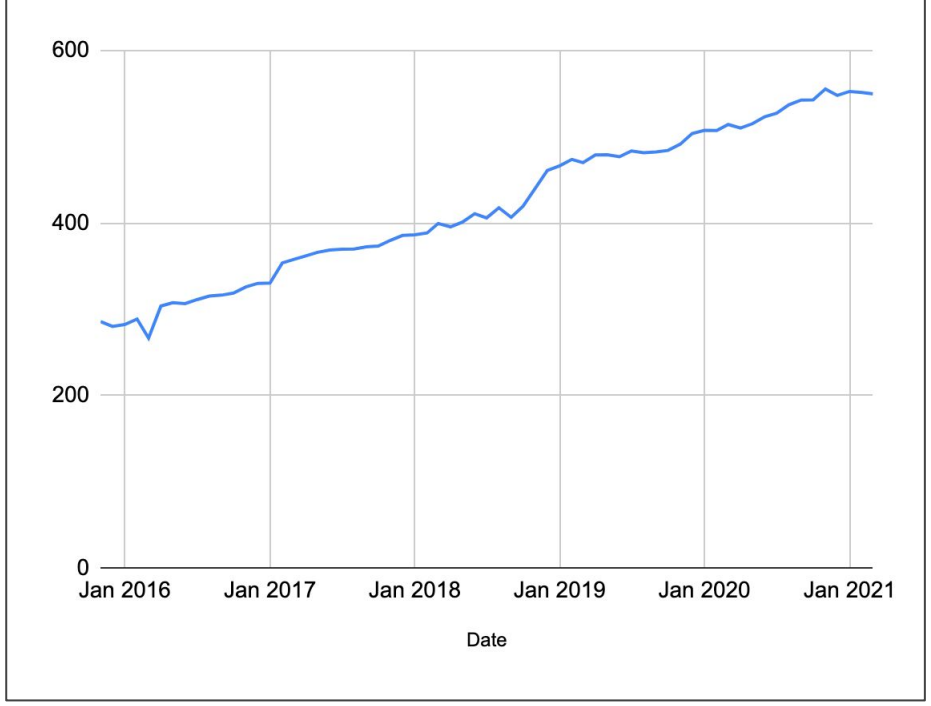
Most popular database



Most loved database



PostgreSQL popularity



Source: Stack Overflow Developer Survey, 2020

Source: DB-Engines.com, 2021

Feature

Versione ridotta

- Replica streaming nativa, logica e fisica, sincrona ed asincrona
- Continuous backup e point in time recovery
- Partizionamento dichiarativo
- Supporto JSON
- Estensioni (e.g. PostGIS)
- Query parallele
- INSERT .. ON CONFLICT DO UPDATE
- Supporto TLS e autenticazione con certificati
- SQL Standard (2016)
- DDL transazionali
- Ricerca full text nativa
- Tablespace
- ...

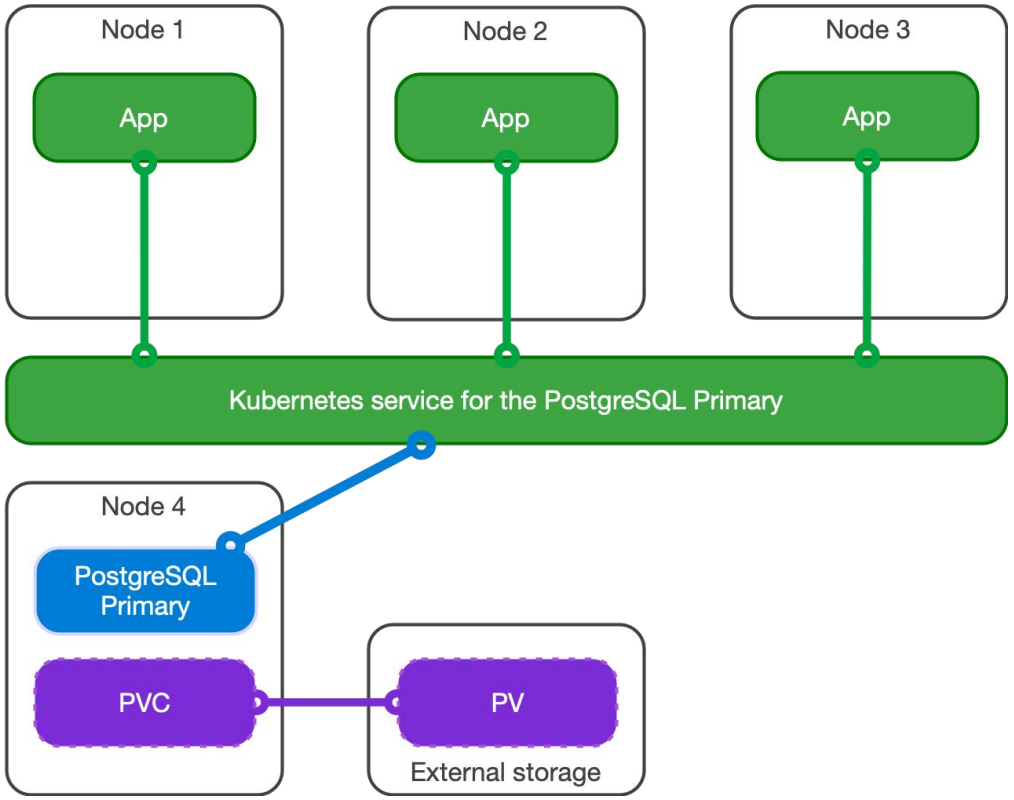
PostgreSQL: architettura di replica

Primario e repliche tramite replica streaming fisica nativa

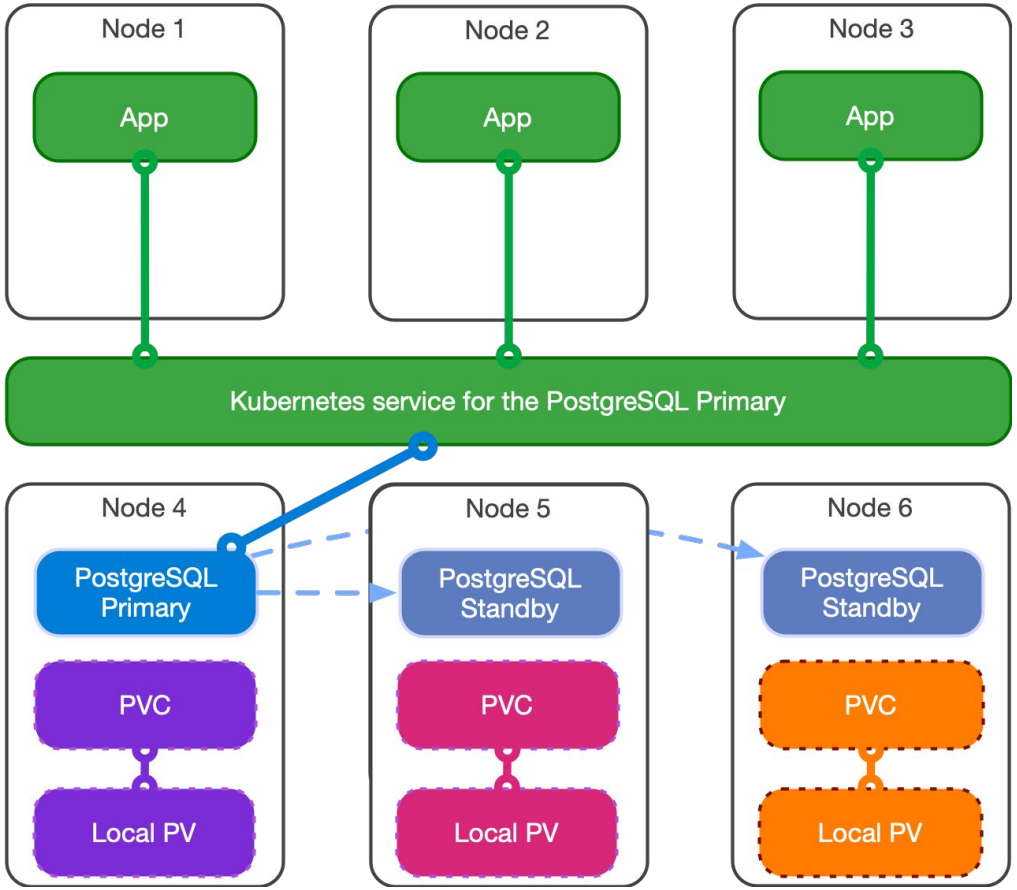
- PostgreSQL supporta in modo nativo l'architettura **primary/standby**
  - Evolutione della **Crash Recovery** e della **Point-In-Time Recovery**
  - Introdotta in PostgreSQL 8.2 con **WAL shipping** e Warm Standby
  - Migliorata in PostgreSQL 9.0 con **WAL streaming** e **Hot Standby** (replica sola lettura)
- Ulteriori miglioramenti:
  - **Replica sincrona**
  - **Replica in cascata**
  - **Replica logica**
- Robusta e affidabile, con ottimi risultati in termini di RPO e RTO

Due approcci a PostgreSQL su Kubernetes

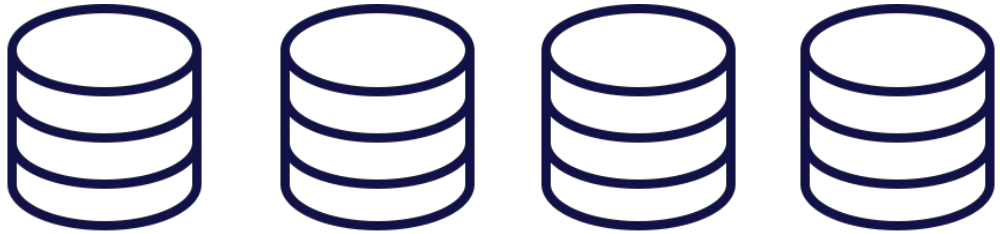
Container (con replica del file system)



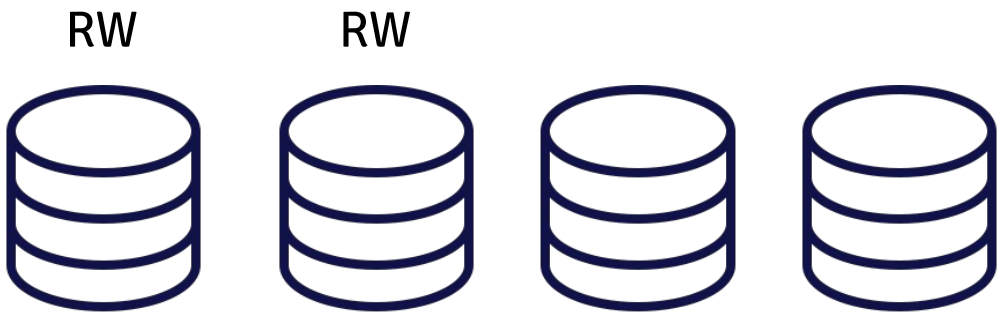
Operatore (con replica PostgreSQL)



Stato richiesto



Stato attuale



Cloud Native PostgreSQL by EDB

Il migliore database open source sul migliore orchestratore di container open source

- Operatore proprietario, con una licenza di prova implicita di 30 giorni
- Operandi:
  - PostgreSQL
  - EDB Postgres Advanced Server
- Documentazione disponibile
  - docs.enterprisedb.io

Perché fare benchmark su PostgreSQL

Metodo scientifico

- Decisioni basate su dati e misurazioni ripetibili
  - “Lo sanno tutti” vs. “Perché le TPS osservate non sono sufficienti”
- Obiettivi:
  - Capacity planning e ottimizzazione dei costi
  - Prestazioni predicibili
- Prestazioni
  - Possono variare per molteplici fattori (hardware, software, network!)
  - Il database è pesantemente influenzato dallo storage (bottleneck)
- **Prima di andare in produzione**

Parte 1
Come è iniziato

Un nuovo prodotto in 2ndQuadrant

L'esplorazione di Kubernetes inizia nel Q3/2019

- **Contesto:** esperienza pluriennale su DevOps/Lean/Agile
- Portare PostgreSQL su Kubernetes
  - Cosa manca dentro PostgreSQL per un esperto Kubernetes?
  - Cosa manca in Kubernetes per un esperto PostgreSQL?
  - **Esplorazione con approccio fail-fast**
- Obiettivo: arrivare in Kubernetes allo stesso livello che abbiamo su PostgreSQL
  - Prima compagnia PostgreSQL a diventare Kubernetes Certified Service Provider
- Per le nostre esperienza, le prestazioni dello storage erano la priorità principale
  - **Possiamo fare funzionare PostgreSQL in un Kubernetes su bare-metal?**

Local Persistent Volumes and PostgreSQL usage in Kubernetes

May 7, 2020 / in Cloud Native, Gabriele's PlanetPostgreSQL / by Gabriele Bartolini

Can I use PostgreSQL in Kubernetes and expect to achieve performance results of the storage that are comparable to traditional installations on bare metal or VMs? In this article I go through the benchmarks we did in our own Private Cloud based on Kubernetes 1.17 to test the performance of local persistent volumes using OpenEBS Local PV.

Parte 2
Storage

Cosa si aspetta un database dallo storage?

In bare metal, VM e container

- Disponibilità
- Scalabilità
- **Prestazioni**
- **Consistenza**
- **Durabilità**

Storage e architetture

Network

- Pattern K8s
- Numerose soluzioni disponibili
- HA tramite replica dello storage
- Storage condiviso
  - Altri workload

Locale

- Antipattern K8s
- Accesso diretto allo storage del SO
  - Sul nodo worker
- **Richieste degli utenti PostgreSQL**
  - **Architettura shared nothing**
  - Prestazioni
  - Predicibilità
  - HA tramite replica Postgres

PostgreSQL e lo storage (versione semplificata)

Dove lo storage diventa critico per PostgreSQL

- **Write Ahead Log (WAL)**, in passato xlog
  - Scritture sequenziali e fsync
- **Scrittura degli shared buffers**
  - Fatta dal checkpoint, dal bgwriter, o dal backend
  - Scritture random
- **Letture di pagine**
  - Letture random
- **Table scan**
  - Letture sequenziali

Cosa misurare

Metriche per la valutazione di uno storage

- Throughput:
  - Sequential read
  - Sequential write
  - Random read
  - Random write
- IOPS:
  - Limiti sul cloud

Come misurare

fio: Flexible I/O

- Il tool che abbiamo scelto per le misure
  - In passato abbiamo utilizzato bonnie++
- Simula numerosi tipi di I/O
- Configurable
- Multi-piattaforma
  - Utilizzabile anche su Kubernetes
- Open Source
  - GNU GPL 2
  - https://fio.readthedocs.io/

Se lo storage è lento
il database sarà lento

Parte 3
Database

Workload PostgreSQL

Tipi di carico di lavoro

- In-Memory
  - Il database entra interamente in RAM
  - Limitato da velocità di CPU e RAM
- OnLine Transactional Processing (OLTP)
  - Molte piccole transazioni concorrenti
  - Mix di inserimenti, letture e aggiornamenti
- OnLine Analytical Processing (OLAP)
  - Poche query complesse, principalmente in lettura, su dati storici
  - Usato per reportistica e business intelligence

Cosa misurare

Quali metriche usare per valutare le prestazioni del database?

- Ci siamo concentrati su:
  - **Carico di lavoro OLTP**
  - **Dimensione del database superiore alla RAM**
    - Questo ci costringe a lavorare sul disco
- Metrica scelta:
  - **Transazioni al secondo (TPS)**

Come misurare

pgbench

- Lo strumento che abbiamo scelto per il benchmark del database
- Parte del core di PostgreSQL
- Simula un carico di lavoro *TPC-B like* (OLTP)
- Configurabile
  - scale factor, numero di client e thread, durata, …
- Open Source, TPL
- Supporto per query custom
  - Niente batte l’applicazione reale

Parte 4
cnp-bench

cnp-bench

- Una raccolta di Helm chart
  - Facili da eseguire
  - Facili da personalizzare
  - Facili da riprodurre
- Esecuzione di benchmark per cluster Cloud Native PostgreSQL (CNP)
  - Utilizza fio e pgbench
  - Misurazione di IOPS e TPS
- github.com/EnterpriseDB/cnp-bench

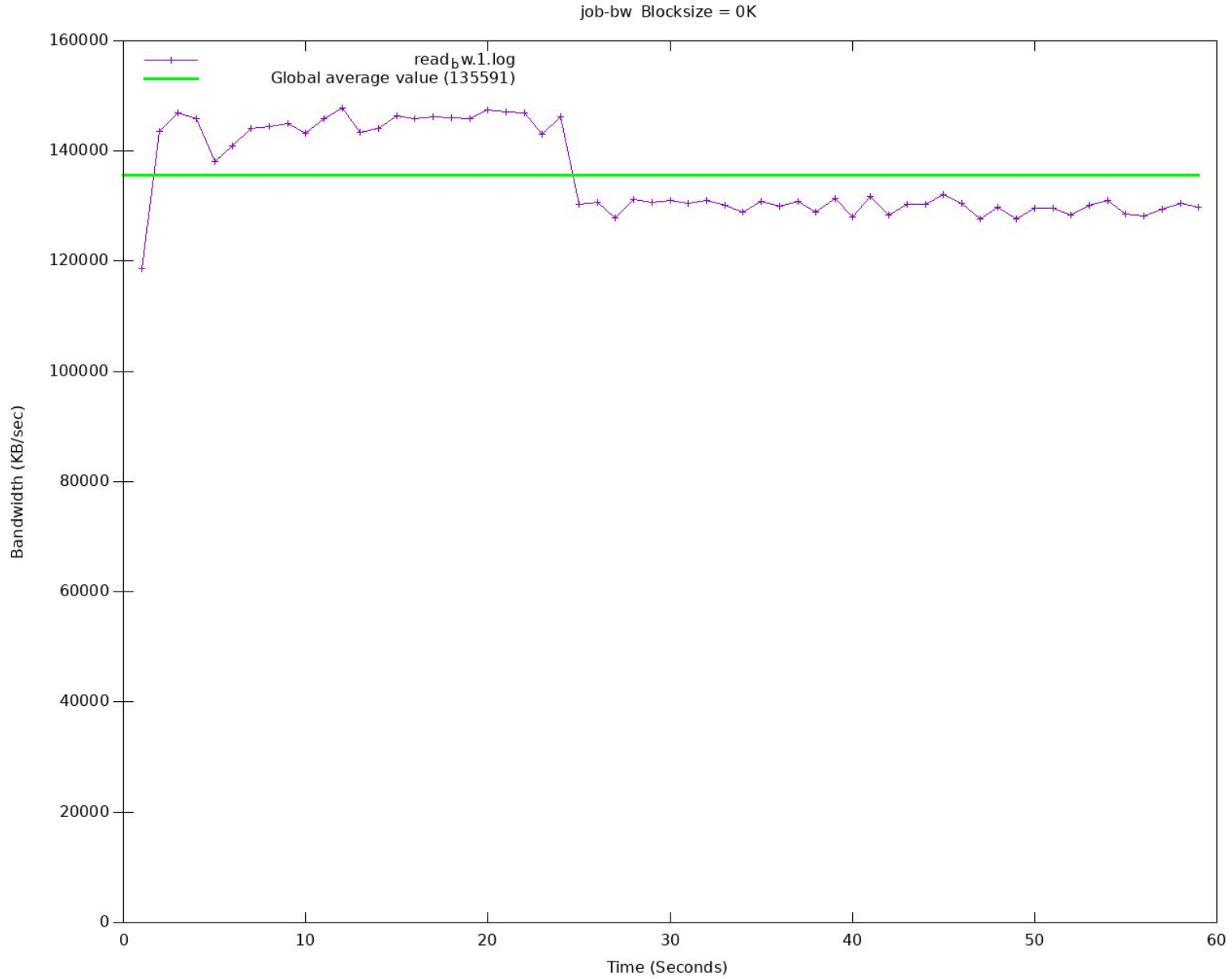
Fio benchmark

- Creazione di un PVC
  - Scegliendo la storage class e la dimensione
- Crea una ConfigMap per il job fio
  - Specifica il tipo di workload da eseguire sul PVC
- Fio è eseguito come deployment
  - Effettua il benchmark
  - Serve i risultati

values.yaml

```
pvcStorageClassName: default
pvcSize: 2Gi
nodeSelector: {}
jobRW: read
jobBs: 8k
jobRuntime: 60
jobTimeBased: 1
jobSize: 1G
jobIodepth: 32
jobDirect: 1
jobIoengine: libaio
jobEndFsync: 1
jobLogAvgMsec: 1000
```

Risultati



pgbench

- Avvia un cluster Cloud Native PostgreSQL
  - Configurabile
  - Impostazioni di PostgreSQL diverse possono cambiare il risultato
- Esegue un job con pgbench
- I log di pgbench contengono i risultati

values.yaml

```
cnpInstances: 1
cnpStorageClass: default
cnpSize: 1Gi
cnpImage: quay.io/enterprisedb/postgresql:13.3
cnpNodeSelector:
  workload: postgresql
cnpPostgreSQLParameters:
  shared_buffers: '512MB'
  maintenance_work_mem: '128MB'
pgbenchNodeSelector:
  workload: pgbench
pgbenchScaleFactor: 1
pgbenchTime: 60
pgbenchClients: 1
pgbenchJobs: 1
pgbenchWarmTime: 0
```

Risultati

```
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 60 s
number of transactions actually processed: 23114
latency average = 2.596 ms
tps = 385.225531 (including connections establishing)
tps = 385.271092 (excluding connections establishing)
```

Parte 5
cnp-bench su AKS

AKS

- Abbiamo eseguito dei test su AKS, con diverse combinazioni di VM dischi
- Le VM non sono definite solo da CPU e RAM
  - Diverse VM hanno limiti diversi per IOPS, larghezza di banda per I/O e rete
- Anche i dischi hanno i loro limiti di IOPS e banda
  - Alcuni dischi permettono di eccedere i limiti per breve tempo (*burst*)

AKS

Standard_E8d_v4

- vCPU: 8
- RAM: 64GB
- Network bandwidth: 4000 Mb/s

Network storage

- P10
- P30
- P80

Disk	Size (GiB)	BW (MiB/s)	IOPS	Burst
P10	100	100	500	Yes
P30	1000	200	5000	No
P80	20000	900	20000	No

Benchmark fio: risultati

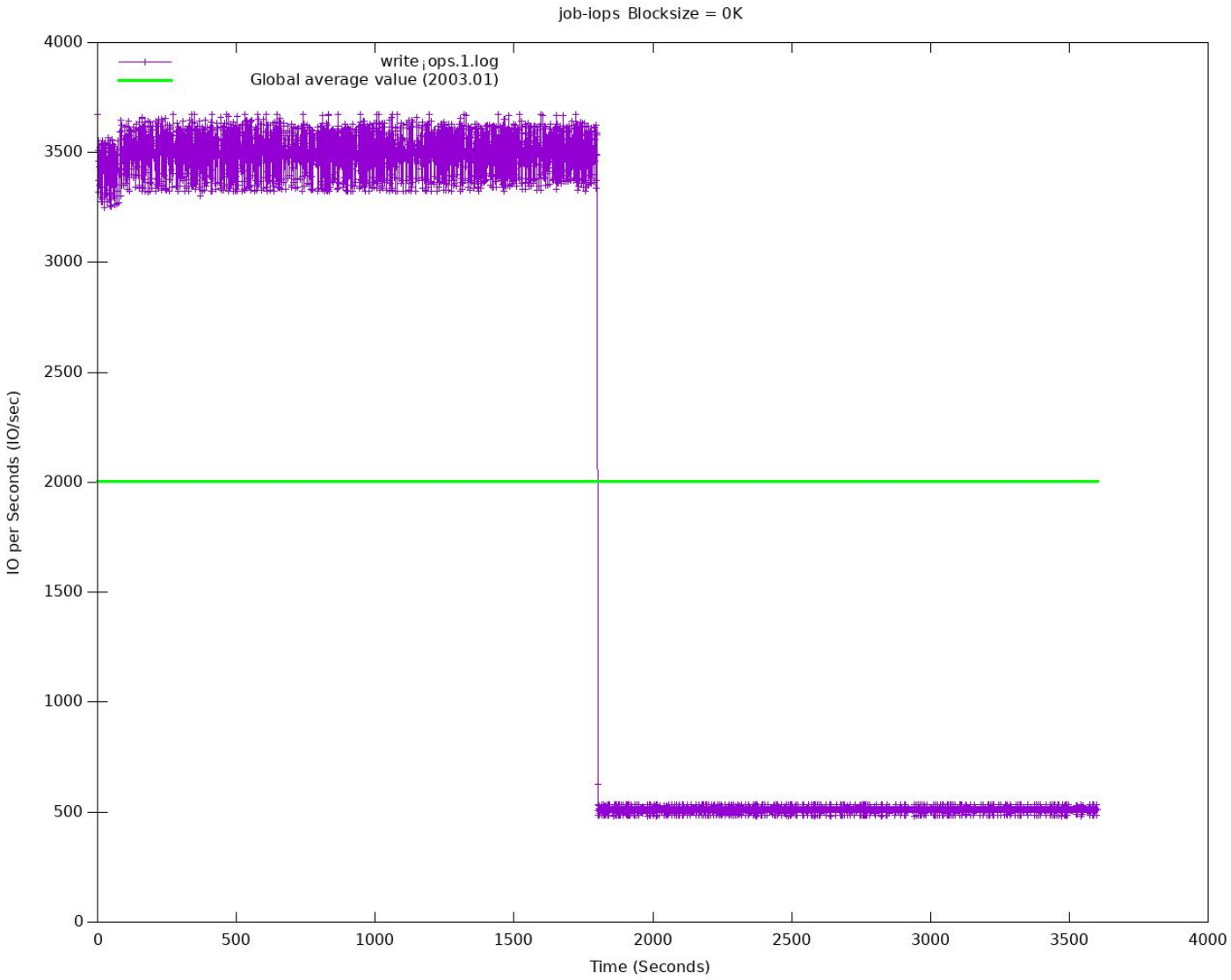
3600s - 8kb block size

Disk	Max BW (MiB/s)	Max IOPS	Seq Read		Seq Write		Rand Read		Rand Write	
			MB/s	IOPS	MB/s	IOPS	MB/s	IOPS	MB/s	IOPS
P10	100	500	**483**	58994	**16.4**	2003	**452**	55137	**15.5**	1960
P30	200	5000	**85.3**	10415	**41.6**	5082	**42.7**	5217	**37.5**	4575
P80	900	20000	**81.3**	9930	**89.1**	10881	**92.6**	11306	**68.2**	9655

Il burst

Scritture sequenziali sul P10

Max BW (MiB/s)	Max IOPS	Seq Write	
		MB/s	IOPS
100	500	**16.4**	2003



La cache di lettura

Disk	BW (MiB/s)	IOPS	Seq Read		Seq Write		Rand Read		Rand Write	
			MB/s	IOPS	MB/s	IOPS	MB/s	IOPS	MB/s	IOPS
P10	100	500	**483**	58994	**16.4**	2003	**452**	55137	**15.5**	1960
P10NC	100	500	**16.3**	1986	**16.4**	2003	**16.3**	1985	**15.5**	1952
P30	200	5000	**85.3**	10415	**41.6**	5082	**42.7**	5217	**37.5**	4575
P30NC	200	5000	**41.7**	5088	**41.6**	5082	**41.7**	5087	**37.5**	4580
P80	900	20000	**81.3**	9930	**89.1**	10881	**92.6**	11306	**68.2**	9655
P80NC	900	20000	**88.3**	10778	**88.2**	10795	**90.7**	11,074	**68.3**	9621

Pgbench: risultati

Durata: 1h

Dimensioni diverse del DB per dischi diversi

Configurazioni di PostgreSQL (comuni):

- shared_buffers: '16GB'
  - 64GB di RAM
- max_wal_size: '15GB'
- checkpoint_timeout: '900s'
- checkpoint_completion_target: '0.9'
- maintenance_work_mem: '2GB'

Disk	DB size (GB)	TPS
P10	71	**1,499**
P30	731	**1,274**
P80	1,490	**1,051**

Work in progress: storage locale

Standard_L8s_v2

CPU: 8

RAM: 64

Throughput:

Max Read BW: 2000 MB/s

Max Read IOPS: 400000

Disks: 1x1.92 TB NVME

Seq Read		Seq Write		Rand Read		Rand Write		DB size (GB)	TPS
MB/s	IOPS	MB/s	IOPS	MB/s	IOPS	MB/s	IOPS		
1,142	139,370	**696**	84,996	**1,070**	130,587	**262**	32,010	732	**2,082**

Conclusioni

Riepilogo degli argomenti

PostgreSQL su Kubernetes? Si può fare!

SCAN ME

- Un metodo per effettuare benchmark di PostgreSQL su Kubernetes
- Degli strumenti open source per il benchmark
- Perché è importante fare benchmark dello storage e del database
- Usare cnp-bench per il benchmark di Cloud Native PostgreSQL:
  - github.com/EnterpriseDB/cnp-bench

Stiamo assumendo!

SCAN ME

- enterprisedb.com/careers
- Numerose posizioni
  - Da remoto, in tutto il mondo
- Nel nostro team:
  - GoLang Developer, Kubernetes
  - Automation engineer, DevOps/Kubernetes