Cloud Native Days Bologna Workshop

23/06/2025

Ввласенко

CC BY SA

EDB POSTGRES AI

PostgreSQL su Kubernetes con CloudNativePG (CNPG)

a cura di
Jonathan Battiato & **Jonathan Gonzalez**

Jonathan Battiato

Member of the **Cloud Native Team** at **EDB**
Linux SysAdmin
PostgreSQL Admin
Kubernetes Admin
DoKC Ambassador

github.com/jbattiato
linkedin.com/in/jonathanbattiato
jonathan.battiato@enterprisedb.com

©EDB 2024 — ALL RIGHTS RESERVED.

Jonathan Gonzalez

Member of the **Cloud Native Team** at **EDB**
CNPG Maintainer
Linux Hacker
Kubernetes Developer
Community Contributor

github.com/sxd
linkedin.com/in/jagonzalezv
jonathan.gonzalez@enterprisedb.com

©EDB 2024 — ALL RIGHTS RESERVED.

Agenda

- Setup ambiente di test sul **laptop**
- Introduzione a **CloudNativePG**
- Installazione del **plugin** di kubectl per CNPG
- Primo **deploy** di un cluster CNPG
- Setup ed esecuzione primo **backup**
- Simulazione di un **incidente**
- Lettura dei **log**
- **Upgrade** dell'Operatore e di PostgreSQL
- **Restore** di un backup
- Bonus: Creazione di un **replica cluster**

Carlo Pelagalli

CC BY SA

Contenuti Utili

Per semplificare la vita a tutti, abbiamo creato alcuni file da condividere.

- **bash_aliases.sh**
  - contiene alcune funzioni per semplificare la connessione ai cluster kubernetes
- **LINKS.md**
  - contiene tutti o quasi i link utili per seguire il corso
- **COMMANDS.md**
  - contiene tutti o quasi i comandi da eseguire durante il corso

https://gist.github.com/sxd/

©EDB 2024 — ALL RIGHTS RESERVED.

Setup ambiente di test

Installare i requisiti di **cnpg-playground**:

- Docker
- Kind
- kubectl
- CNPG Plugin
- git

Poi:

- Impostare i limit con `sysctl`
- Eseguire `setup.sh` script
- Esportare la Kube config
- Impostare il context EU
- Creare gli alias
- Testare la connessione

https://github.com/cloudnative-pg/cnpg-playground/

```
$ sudo sysctl \
  fs.inotify.max_user_watches=524288 \
  fs.inotify.max_user_instances=512

$ cd cnpg-playground
$ bash scripts/setup.sh

$ export
KUBECONFIG=<path-to>/cnpg-playground/k8s/kube-config.ya
ml

$ kubectl config use-context kind-k8s-eu

$ . <path-to>/bash_aliases.sh

$ kubectl get pods -A
$ keu get pods -A
```

CloudNativePG

- Operatore nella **Sandbox** della CNCF
- **Gestisce** il ciclo di vita di PostgreSQL
- Usa la configurazione **Dichiarativa**
- Gira su molte distribuzioni Kubernetes supportate
  - Vanilla
  - OpenShift
  - Cloud Service Providers K8S env
  - kind
  - …

https://cloudnative-pg.io/
https://github.com/cloudnative-pg/cloudnative-pg

Come funziona?

- **Gestisce** gli Operandi
  - container con immagini PostgreSQL
- **Estende** Kubernetes con:
  - **Controllers**
    - Sfrutta il server API di Kubernetes per riconciliare lo stato di Postgres
  - **Custom Resource Definitions**
    - Backups
    - Clusters
    - ImageCatalogs
    - Poolers
    - Databases
    - …

https://cloudnative-pg.io/documentation/current/

Minimum required* YAML definition:

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  storage:
    size: 1Gi
```

* **Convention over configuration** paradigm: all the other parameters are set by default.

CloudNativePG - Funzionalità principali

https://cloudnative-pg.io/documentation/current/#main-features

- **High Availability** and **Self-Healing**
- Support for **local PVCs**
- Managed **services** for **rw** and **ro** workloads
- **Continuous backup** (including snapshots)
- **Point In Time Recovery** (incl. snapshots)
- **Scale up/down** of read-only replicas
- "**Security by default**", including mTLS
- Native **Prometheus exporter**
- **Logging** to stdout in **JSON** format
- **Rolling updates**, incl. **minor Postgres** releases
- **Synchronous** replication
- Online **import** of Postgres **databases**
- Separate **volume** for WALs
- Postgres tablespaces, including temporary
- **Replica clusters** and **distributed topologies**
- Declarative **role management**
- Declarative **hibernation** and **fencing**
- **CNPG-I** - interface to develop CNPG **plugins**
- Connection **pooling**
- **Postgres extensions** (pgvector, PostGIS, …)

©EDB 2024 — ALL RIGHTS RESERVED.

CloudNativePG - Ultima release: 1.26

https://cloudnative-pg.io/documentation/current/release_notes/v1.26/

Features

- Declarative **Offline In-Place Major Upgrades** of PostgreSQL
- Declarative management of **extensions** and **schemas**
- Improved **Startup** and **Readiness Probes** for Replicas

Changes

- **CloudNativePG** is now officially a **CNCF project**
- **Deprecation** of **Native Barman Cloud** Support
- **Hibernation** Command Changes (**cnpg plugin**)

©EDB 2024 — ALL RIGHTS RESERVED.

Come si installa?

©EDB 2024 — ALL RIGHTS RESERVED.

CNPG Plugin

1. Installare il plugin di kubectl per CNPG
2. Familiarizzare con i suoi comandi
   - `--help`
3. Installare l'Operatore CNPG v1.25.1
   - Analizzare le risorse create

https://cloudnative-pg.io/documentation/current/kubectl-plugin/#install

```
$ curl -sSfL \

https://github.com/cloudnative-pg/cloudnative-pg/raw/main/hack/install-cnpg-plugin.sh | \
  sudo sh -s -- -b /usr/local/bin

$ kubectl cnpg --help

$ kubectl cnpg install generate \
  --control-plane \
  --version 1.25.1 \
  | kubectl apply -f - --server-side
```

Primo CNPG Cluster

1. Creare un file YAML contenente la definizione di base di un cluster CNPG
2. Aprire un terminale nuovo per monitorare le risorse
3. Applicare il manifesto del cluster
4. Analizzare e navigare tra le risorse create
    - pods
    - services
    - pvc

https://cloudnative-pg.io/documentation/current/quickstart/#part-3-deploy-a-postgresql-cluster

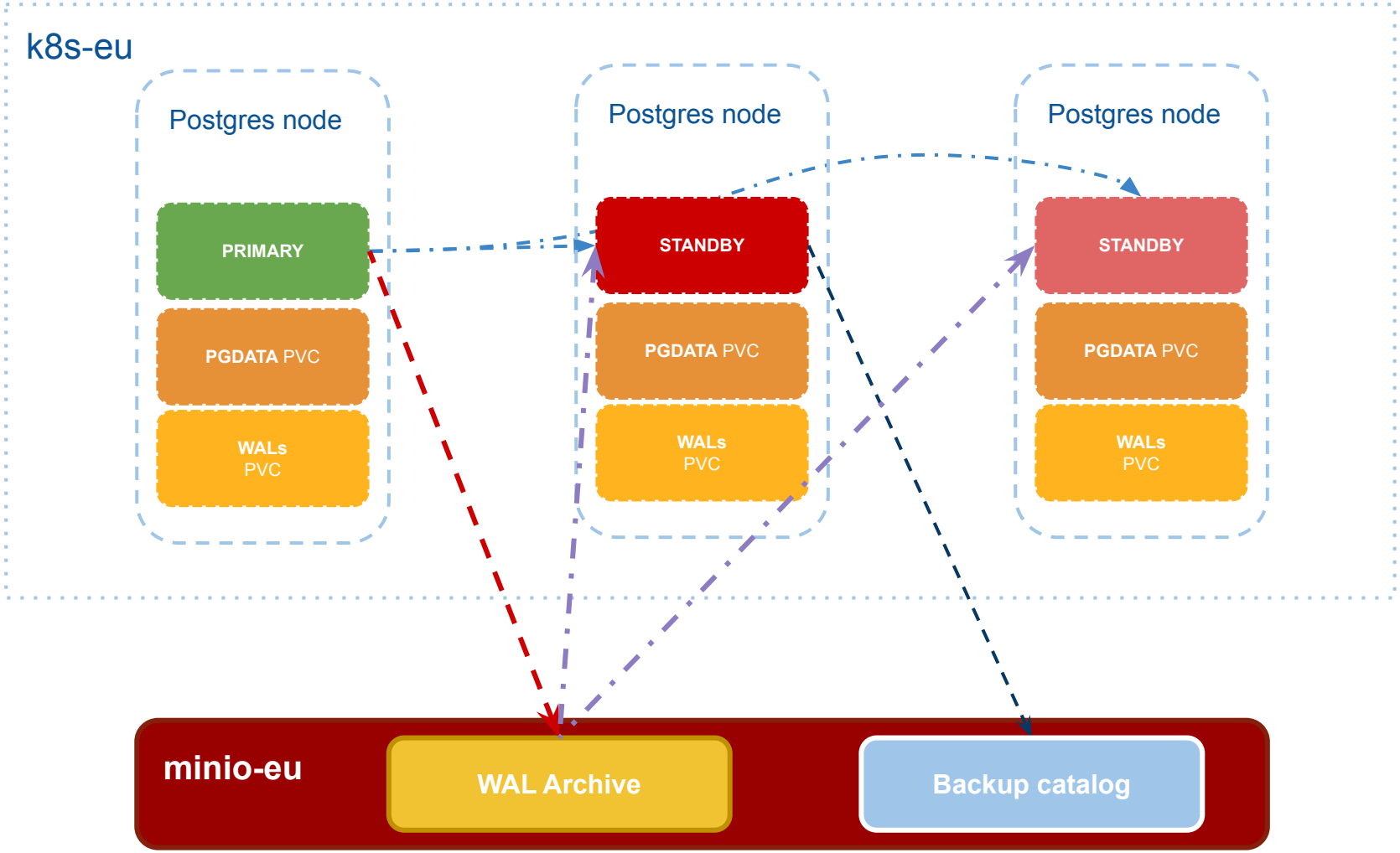
```
$ cat <<EOF > ./cluster-example.yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  storage:
    size: 1Gi
EOF

$ kubectl get pods -w

$ kubectl apply -f cluster-example.yaml

$ kubectl get clusters,pods,pvc,svc,ep
```

Architettura Cluster CNPG



©EDB 2024 — ALL RIGHTS RESERVED.

Come si crea un backup?

©EDB 2024 — ALL RIGHTS RESERVED.

Backup

Metodi:

- `barmanObjectStore`
  - Usa il Barman Cloud presente nelle immagini PostgreSQL per gestire i backup
- `volumeSnapshots`
  - Usa le API native di Kubernetes per fare gli snapshot dei volumi
- `plugin`
  - Usa i plugin creati con l'interfaccia CNPG-i (es: Plugin Barman Cloud)

https://cloudnative-pg.io/documentation/current/backup/

©EDB 2024 — ALL RIGHTS RESERVED.

Primo CNPG Backup

1. Creare un file YAML contenente la definizione di un cluster con backup
2. Monitorare le risorse in un terminale separato
3. Applicare il manifesto
4. Analizzare il cluster creato
    - Usare il comando `status` del plugin

```
$ cat <<EOF > ./cluster-example-backup.yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example-backup
spec:
  instances: 3
  storage:
    size: 1Gi
  backup:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio-eu:9000
      s3Credentials:
        accessKeyId:
          name: minio-eu
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio-eu
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip
EOF

$ kubectl apply -f cluster-example-backup.yaml

$ kubectl cnpg status cluster-example-backup
```

Primo CNPG Backup

1. Generare dati nel DB
   - Accedere al DB `app` con il plugin
   - Creare la tabella `numbers`
   - Inserire 1.000.000 di righe
2. Creare il primo backup
   - Usare il comando `backup` del plugin
3. Analizzare la risorsa backup
4. Analizzare il cluster con il plugin
   - Controllare com'è cambiato il `First Point of Recoverability`

https://cloudnative-pg.io/documentation/current/kubectl-plugin/#requesting-a-new-physical-backup

```
$ kubectl cnpg psql cluster-example-backup -- app

app=# CREATE TABLE numbers(x int);

app=# INSERT INTO numbers (SELECT
generate_series(1,1000000));

app=# \q

$ kubectl cnpg backup cluster-example-backup

$ kubectl get backup

$ kubectl get backup -o yaml

$ kubectl cnpg status cluster-example-backup
```

Che succede in caso di incidente?

©EDB 2024 — ALL RIGHTS RESERVED.

Simulazione di un fallimento

Differenti approcci:

- Rimozione di un **Pod**
  - Viene eliminato solo il pod che esegue il container Postgres
- Rimozione di un **PVC**
  - Viene eliminato il PVC associato al Pod di un'istanza Postgres, contenente i dati
- Rimozione di un **nodo** Kubernetes (non trattato in questo corso)
  - Viene spento o rimosso dal cluster Kubernetes un nodo su cui è in esecuzione un Pod di Postgres

https://cloudnative-pg.io/documentation/current/failure_modes/#failure-modes_1

©EDB 2024 — ALL RIGHTS RESERVED.

Rimozione Pod

1. Individuare il Pod dell’istanza primaria di Postgres
2. Monitorare le risorse in un terminale separato
3. Eseguire il delete del Pod selezionato
4. Verificare lo stato del cluster col plugin

```
$ kubectl get cluster cluster-example-backup

$ kubectl get pods -w

$ kubectl delete pod cluster-example-backup-#

$ kubectl cnpg status cluster-example-backup
```

Rimozione Pod e PVC

1. Individuare il Pod dell'istanza primaria di Postgres
2. Monitorare le risorse in un terminale separato
3. Eseguire il delete del Pod e del PVC selezionati
4. Verificare lo stato del cluster col plugin

```
$ kubectl get cluster cluster-example-backup

$ kubectl get pods -w

$ kubectl delete pod,pvc cluster-example-backup-#

$ kubectl cnpg status cluster-example-backup
```

https://cloudnative-pg.io/documentation/current/failover/

Leggere i log dei Pod

In caso di problemi al Pod di un’istanza Postgres è sempre bene analizzare i log.

1. Analizzare i log del Pod con `kubectl`
2. Analizzare i log del Cluster con il plugin

```
$ kubectl logs cluster-example-backup-#

$ kubectl cnpg logs cluster cluster-example-backup \
  | kubectl cnpg logs pretty
```

Come si affrontano gli aggiornamenti?

©EDB 2024 — ALL RIGHTS RESERVED.

Rolling Updates dell'Operatore

Due fasi:

1. Upgrade dell'**Operatore**
    - Immagine del container dell'Operatore
    - Aggiornamento delle Custom Resource Definition
2. Upgrade del **instance manager** in esecuzione dentro il Pod dell'istanza Postgres

Due metodi:

1. **Automatico**
    - Riavvio dei Pod e switchover/riavvio del primario
2. **Semi-automatico**
    - Riavvio dei soli standby in automatico
    - Switchover/riavvio del primario **manuale**

https://cloudnative-pg.io/documentation/current/installation_upgrade/#upgrades

©EDB 2024 — ALL RIGHTS RESERVED.

Upgrade Operatore CNPG

Al momento abbiamo l'Operatore CNPG v1.25 in esecuzione.

1. Monitoriamo le risorse in **due** terminali separati
   - Uno per il namespace `cnpg-system`
   - Uno per il namespace del cluster (`default`)
2. Installare la nuova versione con il plugin

```
$ kubectl get pods -n cnpg-system -w

$ kubectl get pods -w

$ kubectl cnpg install generate \
  --control-plane \
  --version 1.26.0 \
  | kubectl apply -f - --server-side
```

E gli aggiornamenti di PostgreSQL?

©EDB 2024 — ALL RIGHTS RESERVED.

Rolling Updates dell'Operando

Due contesti:

1. Upgrade della **minor version**
   - Gestito con il metodo dei **Rolling Updates** come per l'upgrade dell'Operando
   - Semplice cambio di immagine dei container PostgreSQL
2. Upgrade della **MAJOR version**
   - Operazione complessa
   - Differenti approcci

Differenti metodi di **Major Upgrade**:

1. **Offline**
   - Usando Dump/Restore logico dei dati da un'istanza ad un'altra - **raddoppio risorse**
   - Upgrade "sul posto" - sostituzione della data directory corrente con quella nuova
2. **Online**
   - Usando la replica logica nativa di PostgreSQL per replicare i dati da una versione a un'altra - **raddoppio risorse**

https://cloudnative-pg.io/documentation/current/postgres_upgrades/

©EDB 2024 — ALL RIGHTS RESERVED.

Upgrade minor version

1. Avendo un cluster con 3 istanze PostgreSQL alla versione 16.3
2. Monitorare le risorse su un terminale separato
3. Applicare il manifesto e attendere che il cluster sia completamente creato con successo
4. Cambiare il TAG version dell'immagine di PostgreSQL da 16.3 a **16.9** nel manifesto
5. Applicare il manifesto con la nuova immagine e verificare con il plugin lo stato del cluster
6. Ripetere il comando `status` del plugin più volte

```
$ cat <<EOF > ./cluster-example.yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  imageName: ghcr.io/cloudnative-pg/postgresql:16.3
  instances: 3
  storage:
    size: 1Gi
EOF

$ kubectl get pods -w

$ kubectl apply -f ./cluster-example.yaml

$ sed -i 's/16\.3/16\.9/' cluster-example.yaml
$ cat ./cluster-example.yaml

$ kubectl apply -f ./cluster-example.yaml \
  && kubectl cnpg status cluster-example

$ kubectl cnpg status cluster-example
```

Upgrade MAJOR version

1. Usando il cluster precedente con 3 istanze PostgreSQL alla versione 16.9
2. Monitorare le risorse su un terminale separato
3. Cambiare il TAG version dell'immagine di PostgreSQL da 16.9 a **17.5** nel manifesto
4. Applicare il manifesto con la nuova immagine e verificare con il plugin lo stato del cluster
5. Ripetere il comando `status` del plugin più volte

```
$ kubectl get pods -w

$ sed -i 's/16\.9/17\.5/' cluster-example.yaml
$ cat ./cluster-example.yaml

$ kubectl apply -f ./cluster-example.yaml \
  && kubectl cnpg status cluster-example

$ kubectl cnpg status cluster-example
```

Come si recupera un backup?

©EDB 2024 — ALL RIGHTS RESERVED.

Recovery

Metodi:

- `barmanObjectStore`
- `volumeSnapshots`
- `plugin`

In CloudNativePG il recupero di un backup:

- È un metodo di **Bootstrap** di un nuovo cluster
- Non può avvenire nel cluster originale
- L'archiviazione dei WAL del nuovo cluster deve avvenire in un nuovo path dell'object store

https://cloudnative-pg.io/documentation/current/recovery/

©EDB 2024 — ALL RIGHTS RESERVED.

CNPG Recovery

1. Usando il cluster `cluster-example-backup` come origine
2. Creare un manifesto con un cluster che utilizzi il metodo `bootstrap.recovery` e come sorgente un `externalCluster` definito con i parametri di accesso all'object store del `cluster-example-backup`

```
$ cat <<EOF > ./cluster-recovery.yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-recovery
spec:
  instances: 3
  storage:
    size: 1Gi
  bootstrap:
    recovery:
      source: origin
  externalClusters:
  - name: origin
    barmanObjectStore:
      serverName: cluster-example-backup
      destinationPath: s3://backups/
      endpointURL: http://minio-eu:9000
      s3Credentials:
        accessKeyId:
          name: minio-eu
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio-eu
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip
EOF
```

CNPG Recovery

1. Monitorare le risorse in un terminale separato
2. Applicare il manifesto del `cluster-recovery`
3. Controllare il cluster con il plugin
4. Verificare i dati nel DB `app`

```
$ kubectl get pods -w

$ kubectl apply -f ./cluster-recovery.yaml

$ kubectl cnpg status cluster-recovery

$ kubectl cnpg psql cluster-recovery -- app

app=# SELECT COUNT(*) numbers;
```

Che fare in caso il datacenter andasse a fuoco?

©EDB 2024 — ALL RIGHTS RESERVED.

Replica Cluster

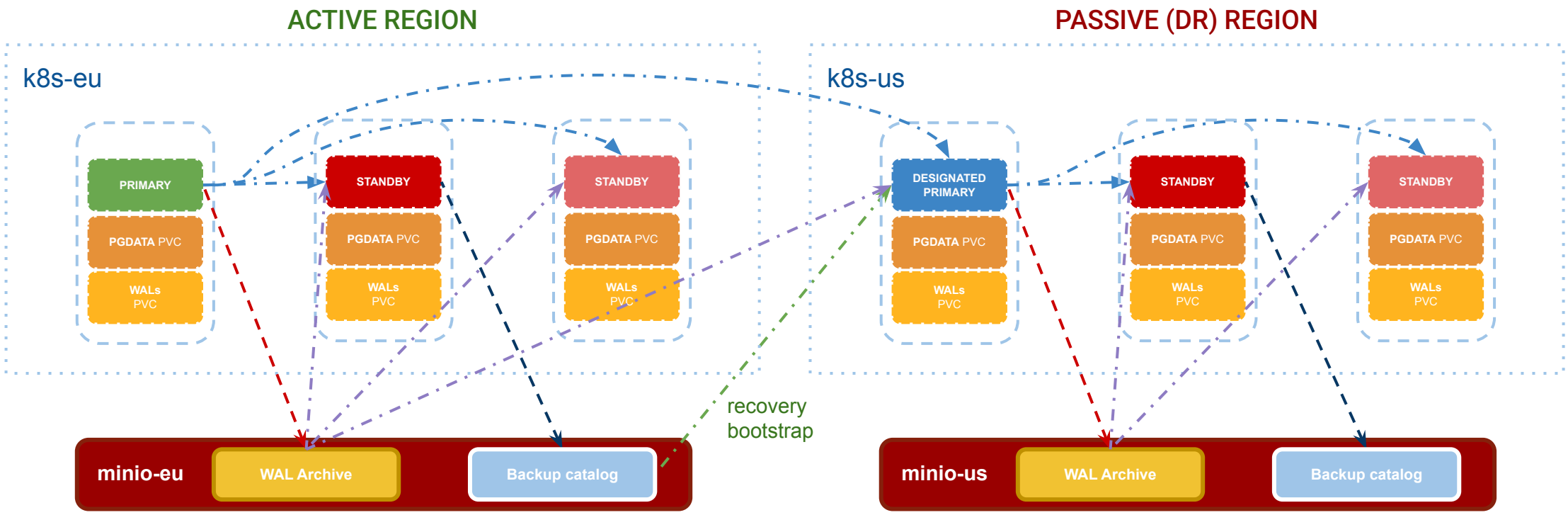
Caratteristiche:

- Due cluster Kubernetes simmetrici in **due** differenti datacenter
  - Il **primo** cluster K8S contiene il **cluster CNPG primario**
  - Il **secondo** cluster K8S contiene il **cluster CNPG in replica**
- Permette la creazione di una **topologia distribuita**
  - Almeno 3 cluster K8S simmetrici che girano su 3 differenti datacenter
  - Il Cluster CNPG primario può essere a turno promosso negli altri datacenter
- Utilizza la **streaming replication** nativa di PostgreSQL
  - Aumenta la resilienza e **riduce il rischio di SPOF** (single point of failure)

https://cloudnative-pg.io/documentation/current/replica_cluster/

©EDB 2024 — ALL RIGHTS RESERVED.

Simulazione Secondo Datacenter (DR)



https://cloudnative-pg.io/documentation/current/replica_cluster/#distributed-topology

©EDB 2024 — ALL RIGHTS RESERVED.

Cluster CNPG in k8s-eu

1. Assicurarsi di usare il context di **k8s-eu** per connettersi al primo cluster K8S
2. Applicare il manifesto del cluster CNPG **pg-eu** nel cluster K8s-eu
3. Verificare lo stato del cluster
4. Eseguire il primo backup

```
$ kubectl config use-context kind-k8s-eu

$ kubectl apply -f
<path-to>/cnpg-playground/demo/yaml/eu/pg-eu-legacy.yam
l

$ kubectl cnpg status pg-eu

$ kubectl cnpg backup pg-eu
```

Cluster k8s-us

1. Assicurarsi di usare il contesto **k8s-us** per connettersi al secondo cluster K8S
2. Verificare di essere nel cluster giusto
3. Installare l'Operatore CNPG anche qui
4. Verificare l'installazione esplorando le risorse create

```
$ ./scripts/info.sh

$ kubectl config use-context kind-k8s-us

$ kubectl get pods -n cnpg-system
$ kubectl config current-context

$ kubectl cnpg install generate \
  --control-plane | \
  kubectl apply -f - --server-side

$ kubectl get deployment -n cnpg-system

$ kubectl get crd | grep cnpg
```

Cluster CNPG in k8s-us

1. Monitorare le risorse in un terminale separato
2. Applicare il manifesto del cluster CNPG **pg-us** nel cluster K8s-us
3. Verificare lo stato del cluster (designated primary)
4. Analizzare la definizione del cluster e compararla con quella del cluster pg-eu

```
$ kubectl get pods -w

$ kubectl apply -f
<path-to>/cnpg-playground/demo/yaml/us/pg-us-legacy.yam
l

$ kubectl cnpg status pg-us

$ kubectl get cluster pg-us -o yaml

$ kubectl --context kind-k8s-eu \
  get cluster pg-eu \
  -o yaml
```

Switchover

- Monitorare le risorse su più terminali separati per entrambe le regioni (cluster k8s)
- Modificare la risorsa cluster di **pg-eu** usando il context k8s-eu
    - Cambiare il valore del campo `spec.replica.primary` da `pu-eu` a `pg-us`
- Salvare le modifiche ed uscire
- Recuperare il Demotion Token dallo status del cluster pg-eu

```
$ kubectl --context kind-k8s-eu get pods -w

$ kubectl --context kind-k8s-us get pods -w

$ kubectl --context kind-k8s-eu edit cluster pg-eu

$ kubectl --context kind-k8s-eu get cluster pg-eu \
  -o jsonpath='{.status.demotionToken}'

$ kubectl --context kind-k8s-us edit cluster pg-us
```

Switchover

- Modificare la risorsa cluster di **pg-us** usando il context k8s-us, per cambiare contemporaneamente:
  - il parametro `spec.replica.primary` con il valore `pg-us`
  - il parametro `spec.replica.promotionToken` con il token ottenuto nello step precedente dallo status del cluster pg-eu
- Salvare le modifiche e uscire
- Verificare la situazione dei CNPG cluster con il plugin in entrambe le regioni

```
$ kubectl --context kind-k8s-us edit cluster pg-us
```

Grazie!

Restiamo in contatto:

Website: cloudnative-pg.io

Blog: cloudnative-pg.io/blog/

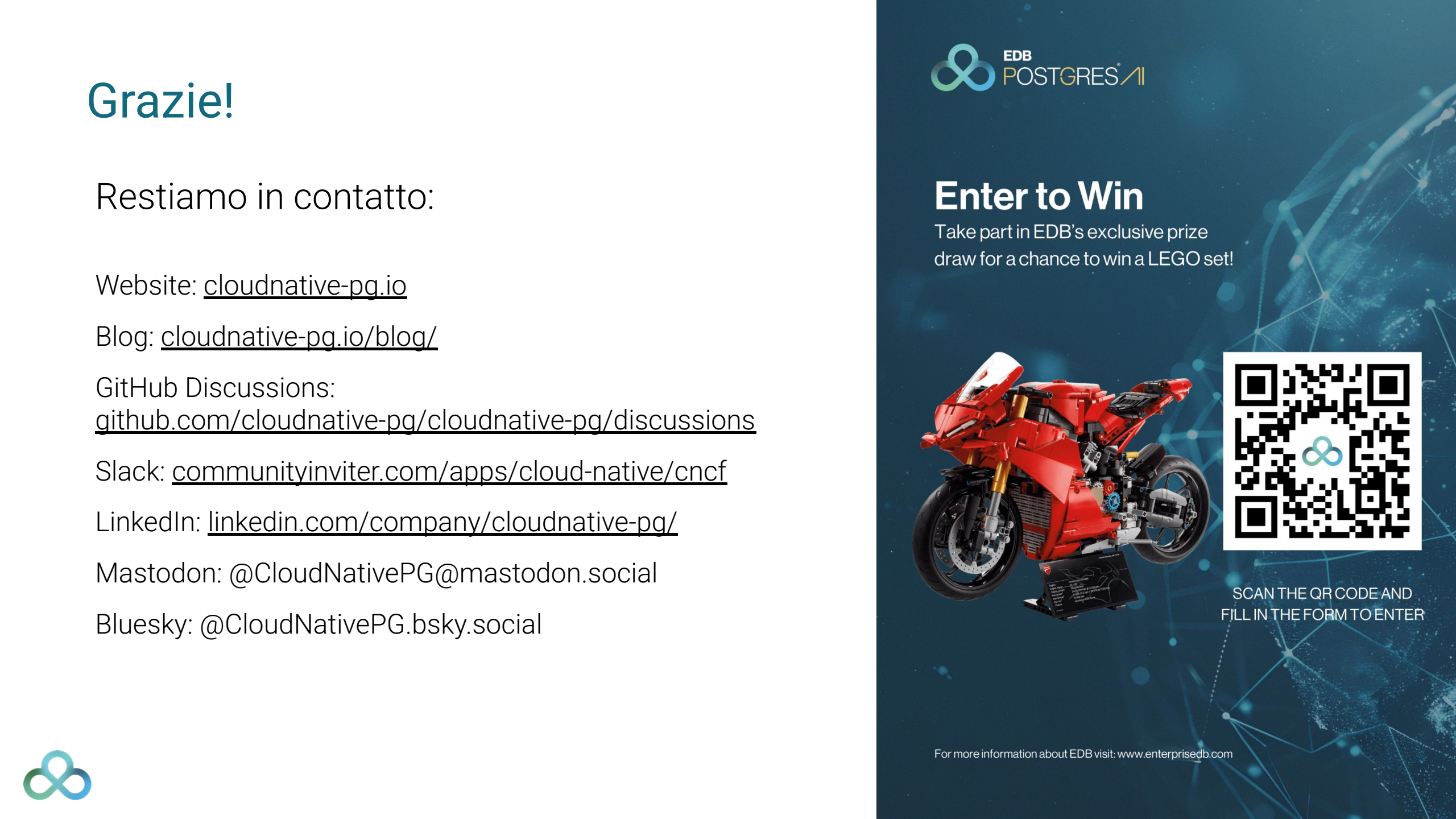
GitHub Discussions: github.com/cloudnative-pg/cloudnative-pg/discussions

Slack: communityinviter.com/apps/cloud-native/cncf

LinkedIn: linkedin.com/company/cloudnative-pg/

Mastodon: @CloudNativePG@mastodon.social

Bluesky: @CloudNativePG.bsky.social



CLOUD NATIVE DAYS ITALY
WORKSHOPS

Feedback Form

©EDB 2024 — ALL RIGHTS RESERVED.