Operating PostgreSQL at Scale With Kubernetes

JONATHAN S. KATZ

MARCH 7, 2019

SCALE17X

crunchy data

About Me

- Director of Communications, Crunchy Data
  - Previously: Engineering leadership in startups
- Longtime PostgreSQL community contributor
  - Advocacy & various committees for PGDG
  - @postgresql + .org content
  - Director, PgUS
  - Conference organization + speaking
- @jkatz05

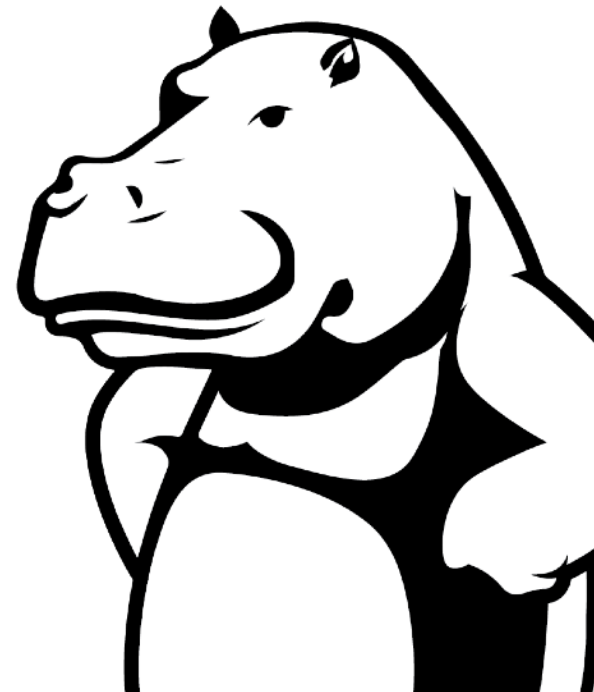
About Crunchy Data

Market Leading Data Security

- Crunchy Certified PostgreSQL is open source and Common Criteria EAL 2+ Certified, with essential security enhancements for enterprise deployment
- Author of the DISA Secure Technology Implementation Guide for PostgreSQL and co-author of CIS PostgreSQL Benchmark. Move ATO from weeks to days!

Cloud Ready Data Management

- Open source, Kubernetes-based solutions proven to scale to 1000s of database instances
- Cloud-agnostic technology provide flexibility on how to deploy databases to public clouds, private clouds, or on-premise technology



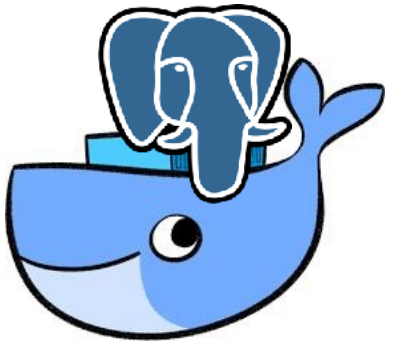
Leader in Open Source Enterprise PostgreSQL

- Developer of essential open source tools for high availability, disaster recovery, and and monitoring for PostgreSQL
- Leading contributor and sponsor of features that enhance stability, security, and performance of PostgreSQL

Outline

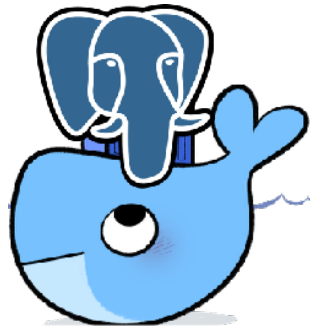
- Containers + PostgreSQL
- Setting up PostgreSQL with Containers
- Operating PostgreSQL at Scale With Kubernetes
- Look Ahead: Trends in the Container World

Containers & PostgreSQL



- Containers provide several advantages to running PostgreSQL:
  - Setup & distribution for developer environments
  - Ease of packaging extensions & minor upgrades
  - Separate out secondary applications (monitoring, administration)
  - Automation and scale for provisioning and creating replicas, backups

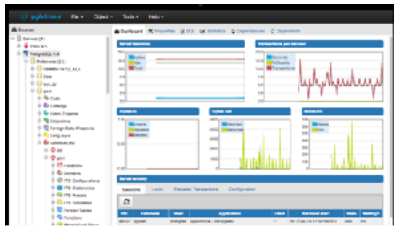
Containers & PostgreSQL



- Containers also introduce several challenges:
  - Administrator needs to understand and select appropriate storage options
  - Configuration for individual database specifications and user access
  - Managing 100s - 1000s of containers requires appropriate orchestration (more on that later)
  - Still a database within the container; standard DBA tuning applies
- However, these are challenges you will find in most database environments

Getting Started With Containers & PostgreSQL

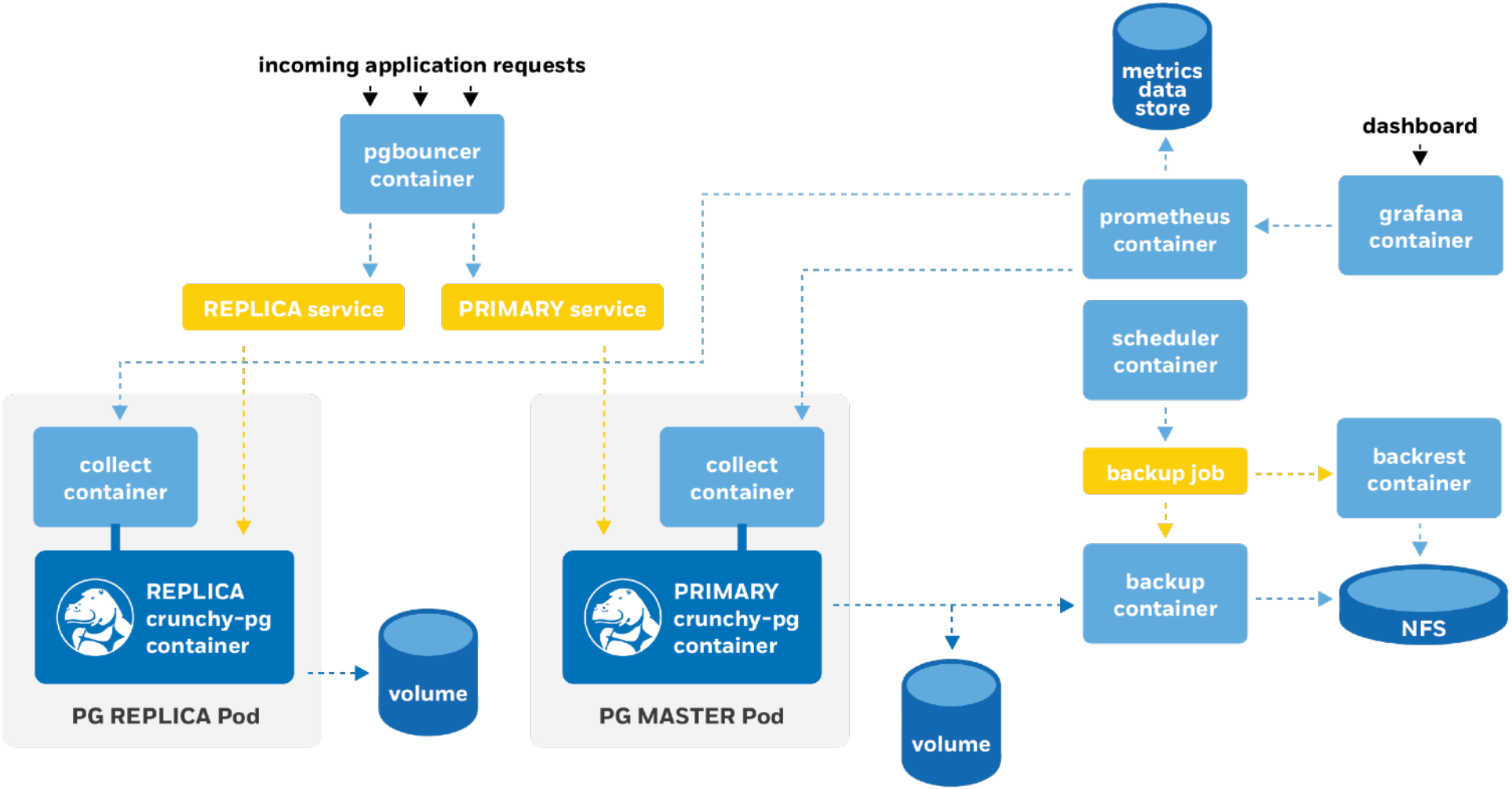
- We will use the Crunchy Container Suite
  - **PostgreSQL (+ PostGIS)**: our favorite database; option to add our favorite geospatial extension
  - **pgpool** + **pgbouncer**: connection pooling, load balancing
  - **pgBackRest**: terabyte-scale disaster recovery management
  - **Monitoring**: pgmonitor
  - **pgadmin4**: UX-driven management
- Open source!
  - Apache 2.0 license
  - Support for Docker 1.12+, Kubernetes 1.5+
  - Actively maintained and updated

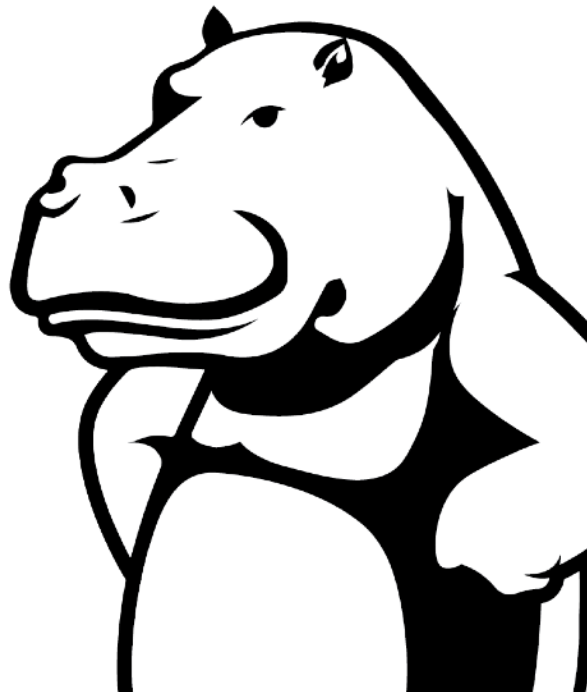


https://github.com/CrunchyData/crunchy-containers

crunchy data

Getting Started With Containers & PostgreSQL





Demo: Creating & Working With Containerized PostgreSQL

```
mkdir postgres && cd postgres

docker volume create --driver local --name=pgvolume
docker network create --driver bridge pgnetwork

cat << EOF > pg-env.list
PG_MODE=primary
PG_PRIMARY_USER=postgres
PG_PRIMARY_PASSWORD=password
PG_DATABASE=whales
PG_USER=jkatz
PG_PASSWORD=password
PG_ROOT_PASSWORD=password
PG_PRIMARY_PORT=5432
PG_LOCALE=en_US.utf8
PGMONITOR_PASSWORD=monitorpassword
EOF

docker run --publish 5432:5432 \
  --volume=pgvolume:/pgdata \
  --env-file=pg-env.list \
  --name="postgres" \
  --hostname="postgres" \
  --network="pgnetwork" \
  --detach \
crunchydata/crunchy-postgres:centos7-11.2-2.3.1
```

Demo: Adding in pgadmin4

```
docker volume create --driver local --name=pga4volume

cat << EOF > pgadmin4-env.list
PGADMIN_SETUP_EMAIL=jonathan.katz@crunchydata.com
PGADMIN_SETUP_PASSWORD=securepassword
SERVER_PORT=5050
EOF

docker run --publish 5050:5050 \
  --volume=pga4volume:/var/lib/pgadmin \
  --env-file=pgadmin4-env.list \
  --name="pgadmin4" \
  --hostname="pgadmin4" \
  --network="pgnetwork" \
  --detach \
crunchydata/crunchy-pgadmin4:centos7-11.2-2.3.1
```

Demo: Adding Monitoring

1. Set up the metric collector

```
cat << EOF > collect-env.list
DATA_SOURCE_NAME=postgresql://ccp_monitoring:monitorpassword@postgres:5432/postgres?sslmode=disable
EOF

docker run \
    --env-file=collect-env.list \
    --network=pgnetwork \
    --name=collect \
    --hostname=collect \
    --detach crunchydata/crunchy-collect:centos7-11.2-2.3.1
```

2. Set up prometheus to store metrics

```
mkdir prometheus

cat << EOF > prometheus-env.list
COLLECT_HOST=collect
SCRAPE_INTERVAL=5s
SCRAPE_TIMEOUT=5s
EOF

docker run \
    --publish 9090:9090 \
    --env-file=prometheus-env.list \
    --volume `pwd`/prometheus:/data \
    --network=pgnetwork \
    --name=prometheus \
    --hostname=prometheus \
    --detach crunchydata/crunchy-prometheus:centos7-11.2-2.3.1
```

3. Set up grafana to visualize

```
mkdir grafana

cat << EOF > grafana-env.list
ADMIN_USER=jkatz
ADMIN_PASS=password
PROM_HOST=prometheus
PROM_PORT=9090
EOF

docker run \
    --publish 3000:3000 \
    --env-file=grafana-env.list \
    --volume `pwd`/grafana:/data \
    --network=pgnetwork \
    --name=grafana \
    --hostname=grafana \
    --detach crunchydata/crunchy-grafana:centos7-11.2-2.3.1
```

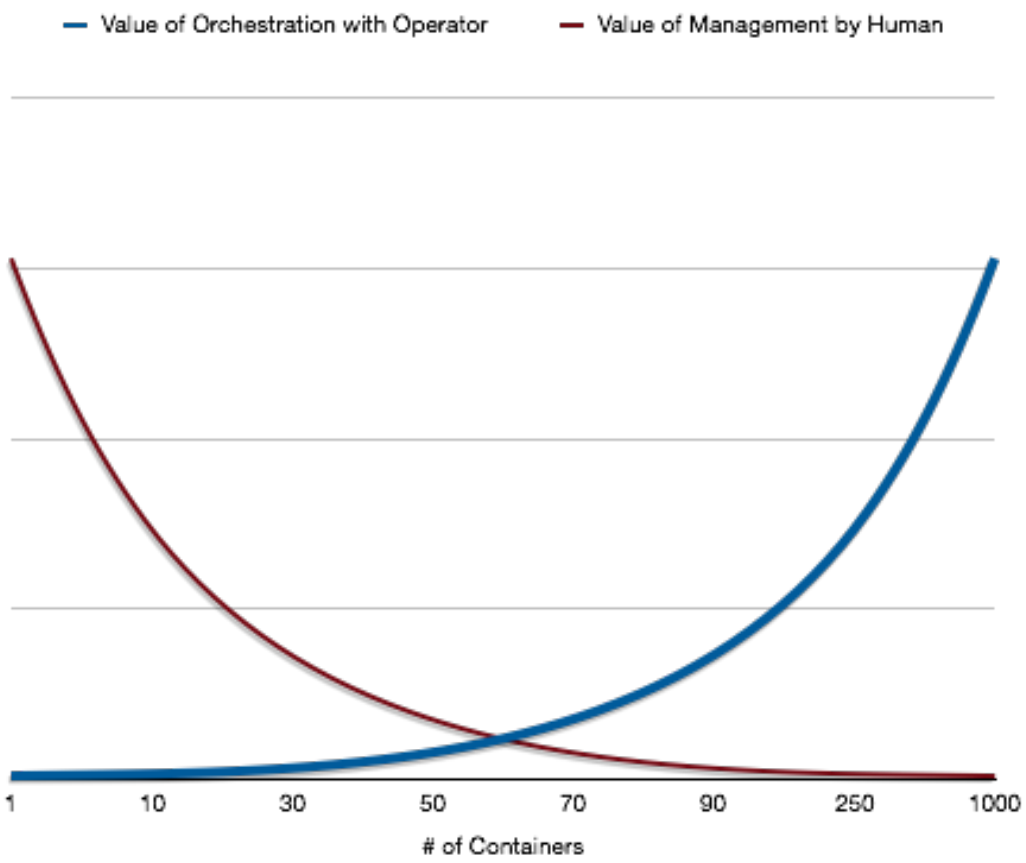
Running PostgreSQL on Kubernetes.

At Scale.

crunchy data

When to Use Kubernetes with PostgreSQL

- Value of Kubernetes increases exponentially as number of containers increases
- Running databases on Kubernetes requires more specialized knowledge than running non-stateful applications
    - What happens to your data after a pod goes down?



crunchy data

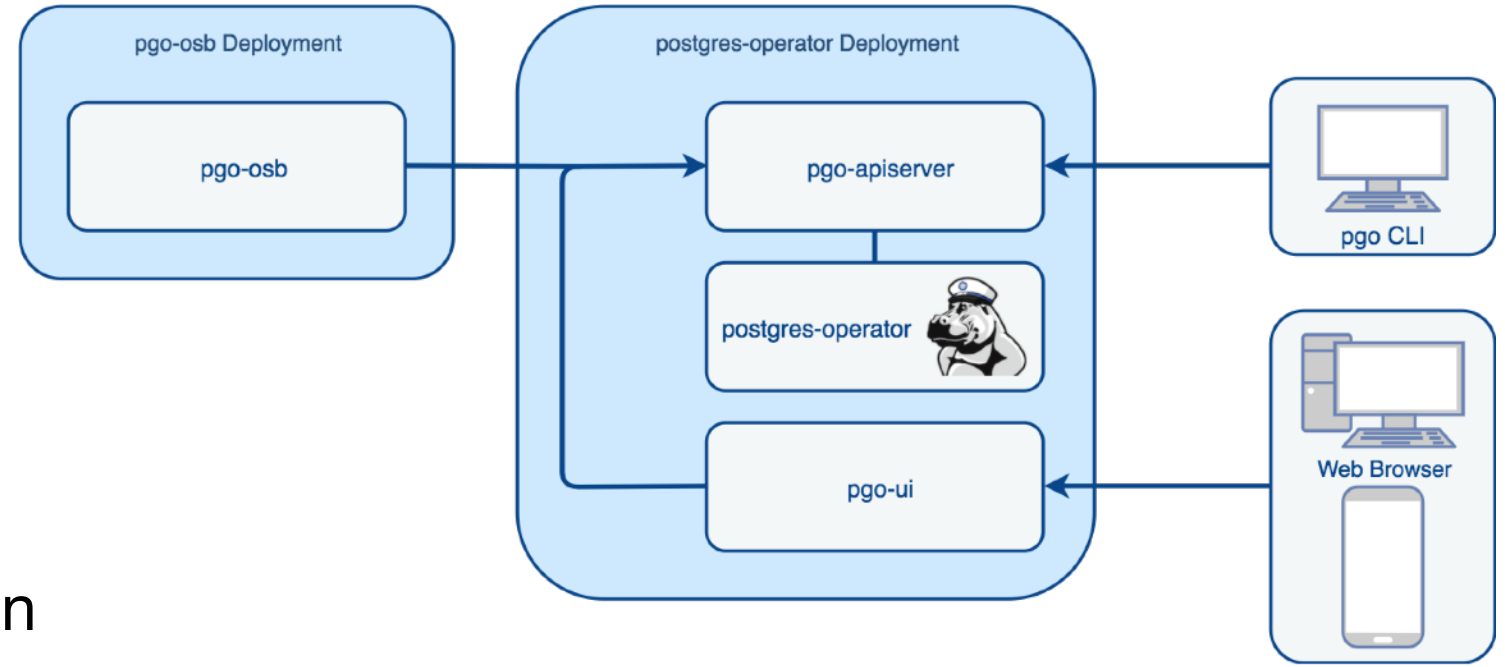
Crunchy PostgreSQL Operator

- PostgreSQL Operator GA: March, 2017
- Allows an administrator to run PostgreSQL-specific commands to manage database clusters, including:
  - Creating / Deleting a cluster (your own DBaaS)
  - Scaling up / down replicas
  - High-Availability
  - Apply user policies to PostgreSQL instances
  - Managing backup intervals and policies
  - Define what container resources to use (RAM, CPU, etc.)
  - Upgrade management
  - Smart pod deployments to nodes
- REST API

https://github.com/CrunchyData/postgres-operator

Crunchy PostgreSQL Operator: Architecture

- Utilizes Kubernetes Deployments:
  - Flexibility in storage classes
  - Flexibility in operating environments
    - Node affinity
    - Resource (CPU, RAM) configurations
  - Flexibility in database version runtimes

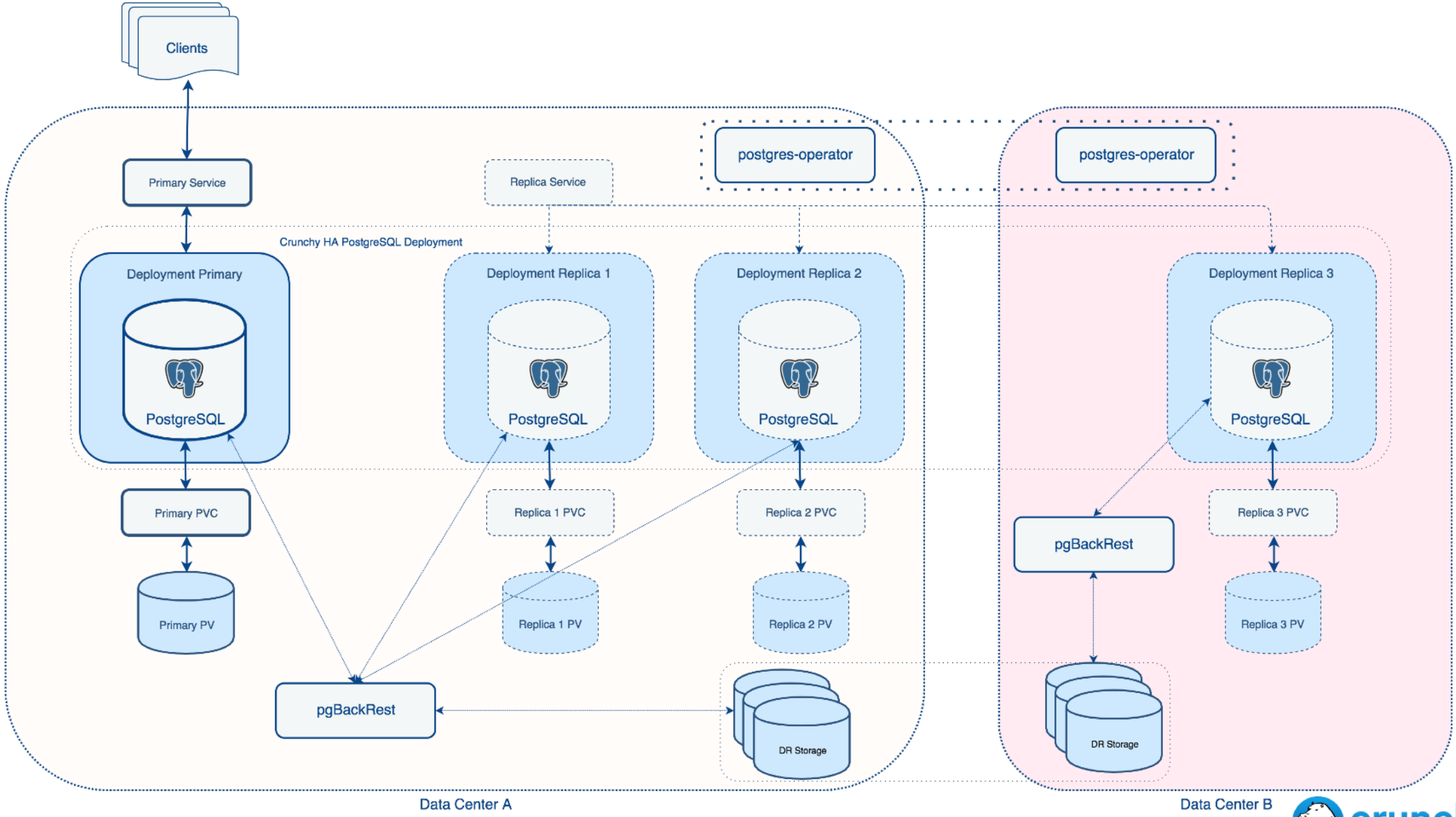


crunchy data

Why Use An Operator With PostgreSQL?

- **Automation**: Complex, multi-step DBA tasks reduced to one-line commands
- **Standardization**: Many customizations, same workflow
- **Ease-of-Use**: Simple CLI
- **Scale**
  - Provision & manage clusters quickly amongst thousands of instances
  - Load balancing, disaster recovery, security policies, deployment specifications
- **Security**: Sandboxed environments, RBAC, mass grant/revoke policies

Why Use An Operator With PostgreSQL?



crunchy data

Demo: Provisioning a Cluster

```
pgo create cluster --autofail --pgbackrest --metrics --replica-count 1 scale17x
pgo show cluster scale17x
```

Demo: Creating a User; Connectivity; Utilization

```
pgo create user jkatz scale17x \
   --password password --managed --selector=name=scale17x
pgo test scale17x
pgo df scale17x
```

Demo: Running Some Tests; Utilization

```
# get the service forward command

# run some pgbench
pgbench -i -s 1 -h localhost -p 5434 userdb
pgbench -c 2 -j 1 -t 128 --progress=1 -h localhost -p 5434 userdb
pgbench -c 2 -j 1 -t 128 -S --progress=1 -h localhost -p 5434 userdb
# Coming in 4.0: pgo benchmark!

pgo df scale17x
```

Demo: Labels; Here is Where We Scale!

```
# labels
pgo label scale17x --label=project=current
pgo create cluster scale18x --labels project=future
pgo create cluster scale19x --labels project=future
pgo show cluster --selector=project=future

pgo create user jkatz --password password --managed --selector=project=future
pgo delete user jkatz --selector=project=future
```

Demo: High-Availability and Horizontal Scaling

```
# It's elastic!
pgo scale scale17x --replica-count=1

# Run some queries on the replica

# HA
pgo failover scale17x --query
pgo failover scale17x --autofail-replace-replica true --target <pod>
pgo test scale17x
```

Demo: Setting Backup Policies

```
# backup policy
pgo create schedule scale17x \
    --schedule="0 0 * * *" \
    --schedule-type=pgbackrest \
    --pgbackrest-backup-type=full

pgo create schedule scale17x \
    --schedule="0 6,12,18 * * *" \
    --schedule-type=pgbackrest \
    --pgbackrest-backup-type=diff

pgo show schedule scale17x
```

Demo: Disaster Strikes!

```
pgo backup scale17x --backup-type=pgbackrest

# log in, do some stuff

# oh no! restore
# can choose to do point-in-time-recovery
# pgo restore scale17x --backup-type=pgbackrest --pitr-target="2019-03-07 17:44:00" -
backup-opts="--type=time"
# or choose to back up up until the last archive
# pgo restore scale17x --backup-type=pgbackrest
```

PostgreSQL & Containers: Looking Ahead

crunchy data

Containerized PostgreSQL: Looking Ahead

- Containers are no longer "new" - orchestration technologies have matured
- Debate with containers + databases: storage & management
  - No different than virtual machines + databases
  - Databases are still databases: need expertise to manage
  - Stateful Sets vs. Deployments
- Federation v2 API opens up new possibilities for high-availability
- Database deployment automation flexibility
  - Deploy your architecture to any number of clouds
- Monitoring: A new frontier

Conclusion

- PostgreSQL + Containers + Kubernetes gives you:
  - Easy-to-setup development environments
  - Your own production database-as-a-service
  - Tools to automate management of over 1000s of instances in short-order

Thank You!

Jonathan S. Katz
jonathan.katz@crunchydata.com
@jkatz05

crunchy data