Running a managed service on Kubernetes and PostgreSQL

What we learned at Timescale

Oleksii Kliukin

Timescale

PostgreSQL Conference Germany, Leipzig

TimescaleDB hypertable

Extend the database with TimescaleDB

```
CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Create a regular table

```
CREATE TABLE IF NOT EXISTS metrics (
   time TIMESTAMP WITHOUT TIME ZONE NOT NULL,
   device_id INT,
   cpu double NULL
);
```

Turn it into a hypertable

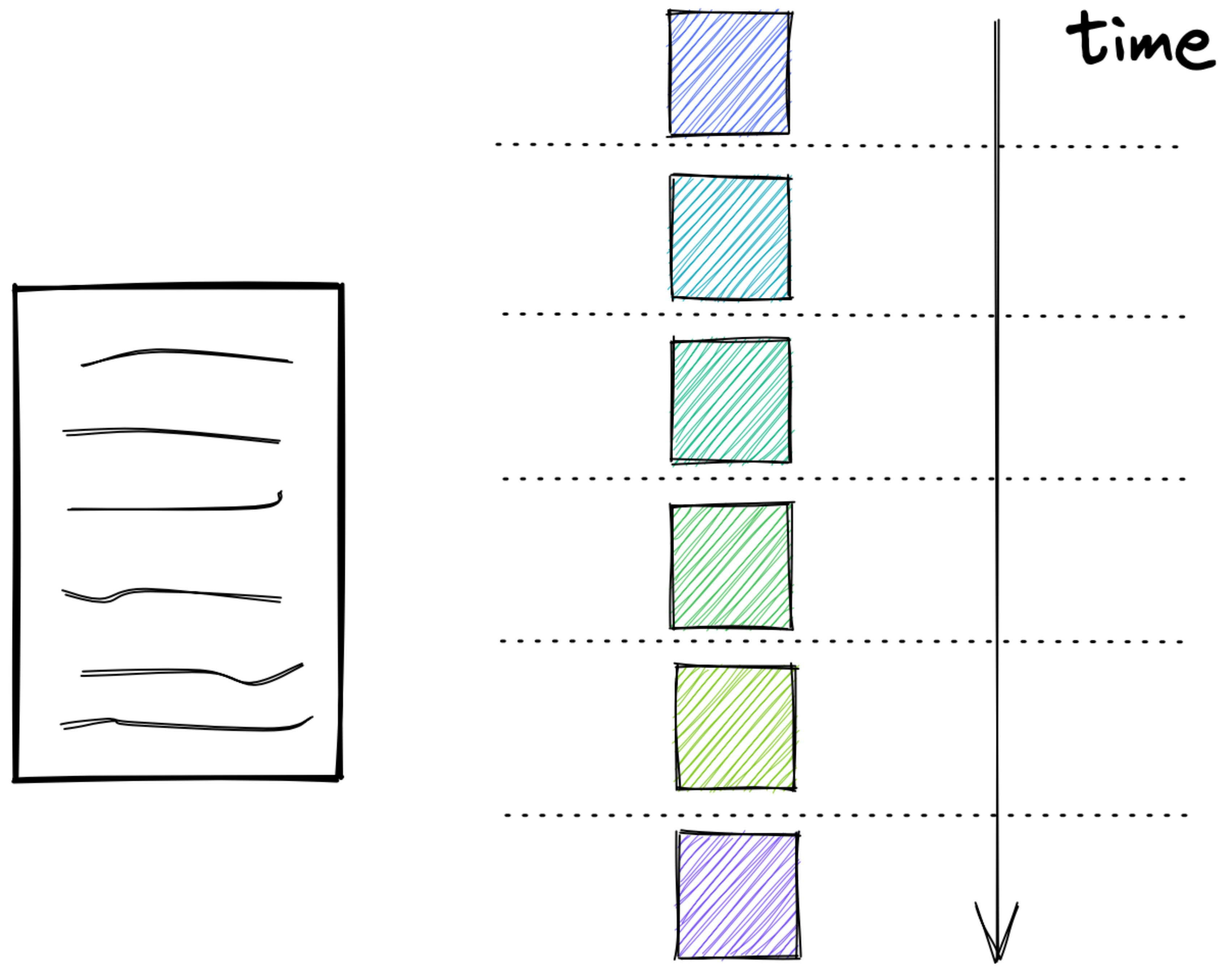
```
SELECT create_hypertable('metrics', 'time');
```

Turn data into hypertable

```
INSERT INTO metrics
SELECT * FROM old_table
```

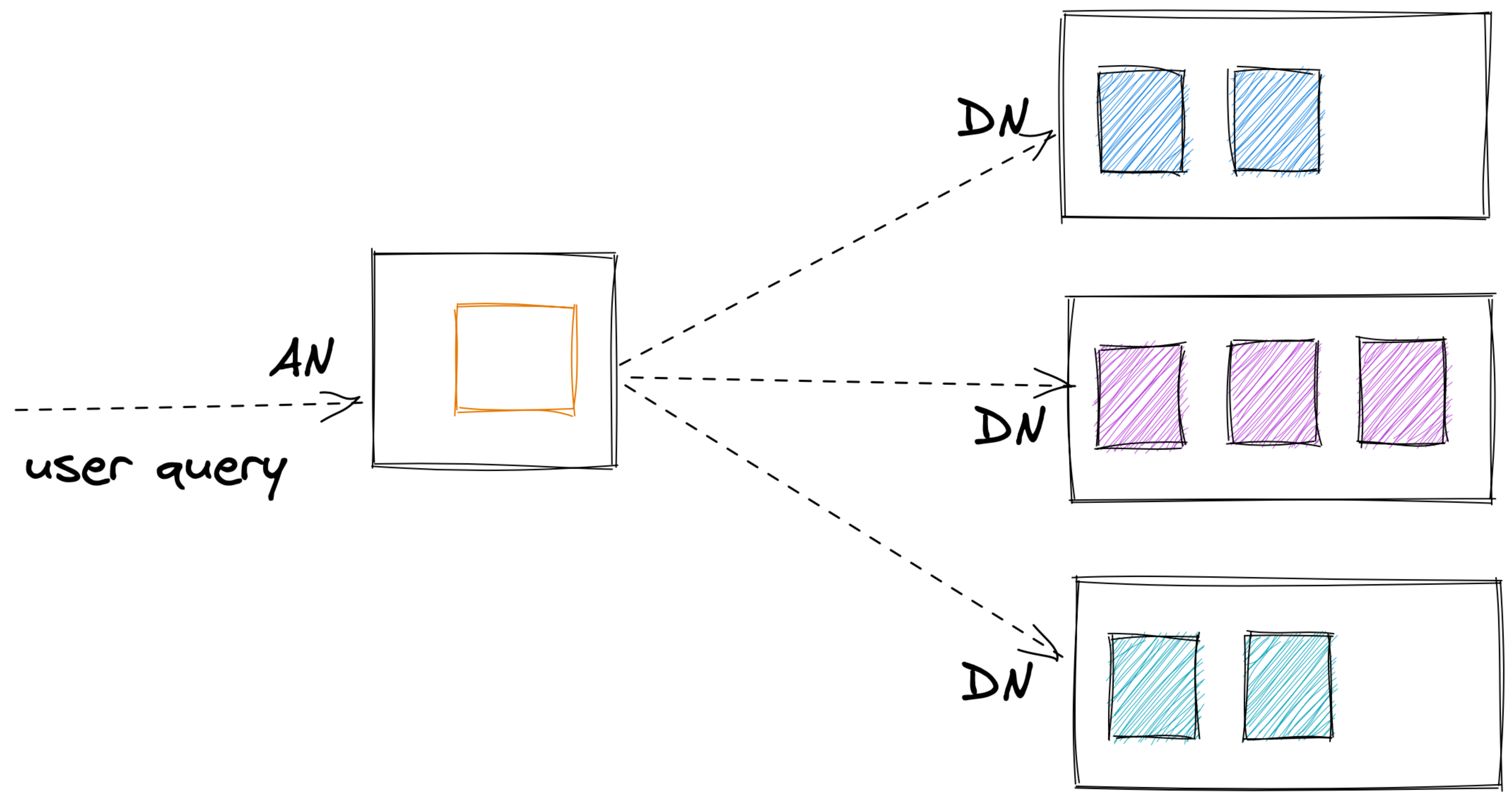
TimescaleDB hypertable chunks

```
SELECT create_hypertable(‘metrics’, ‘time’);
```



Distributed hypertables and multi-node

```
SELECT create_distributed_hypertable(‘metrics’, ‘time’, ‘device_id);
```



There is more to TimescaleDB

- Transparent compression
- Continuous and real-time aggregates
- Data retention policies
- Advanced analytical functions
- Query performance improvements

Agenda

01 Architecture overview 🏗️

02 Challenges 🙃

03 Developer experience 😀

01

Architecture overview

TimescaleCloud

Project
13106_DB

Services
Members
VPC
Support
Billing

```
Usage:        $32  $0
Running:    4 services
Free trial:    45 days
```

Oleksii Kli...
Invitations
Account
Logout
Docs

+ Create service

13106_DB

db-13108 SINGLE-NODE Paused

CPU	RAM	Disk utilization	Region
0.5	2 GB	4% 390 MB of 10 GB	us-east-1

Created 18 days ago

db-22595 SINGLE-NODE Running

CPU	RAM	Disk utilization	Region
0.5	2 GB	3% 270 MB of 10 GB	eu-west-1

Created 4 minutes ago

db-90530 SINGLE-NODE Running

CPU	RAM	Disk utilization	Region
0.5	2 GB	4% 390 MB of 10 GB	eu-central-1

Created 20 days ago

db-multinode MULTI-NODE Running

Nodes		CPU/node	RAM/node	Region
Access nodes	1	1	4 GB	eu-west-1
Data nodes	3	1	4 GB	

Highest disk utilizations

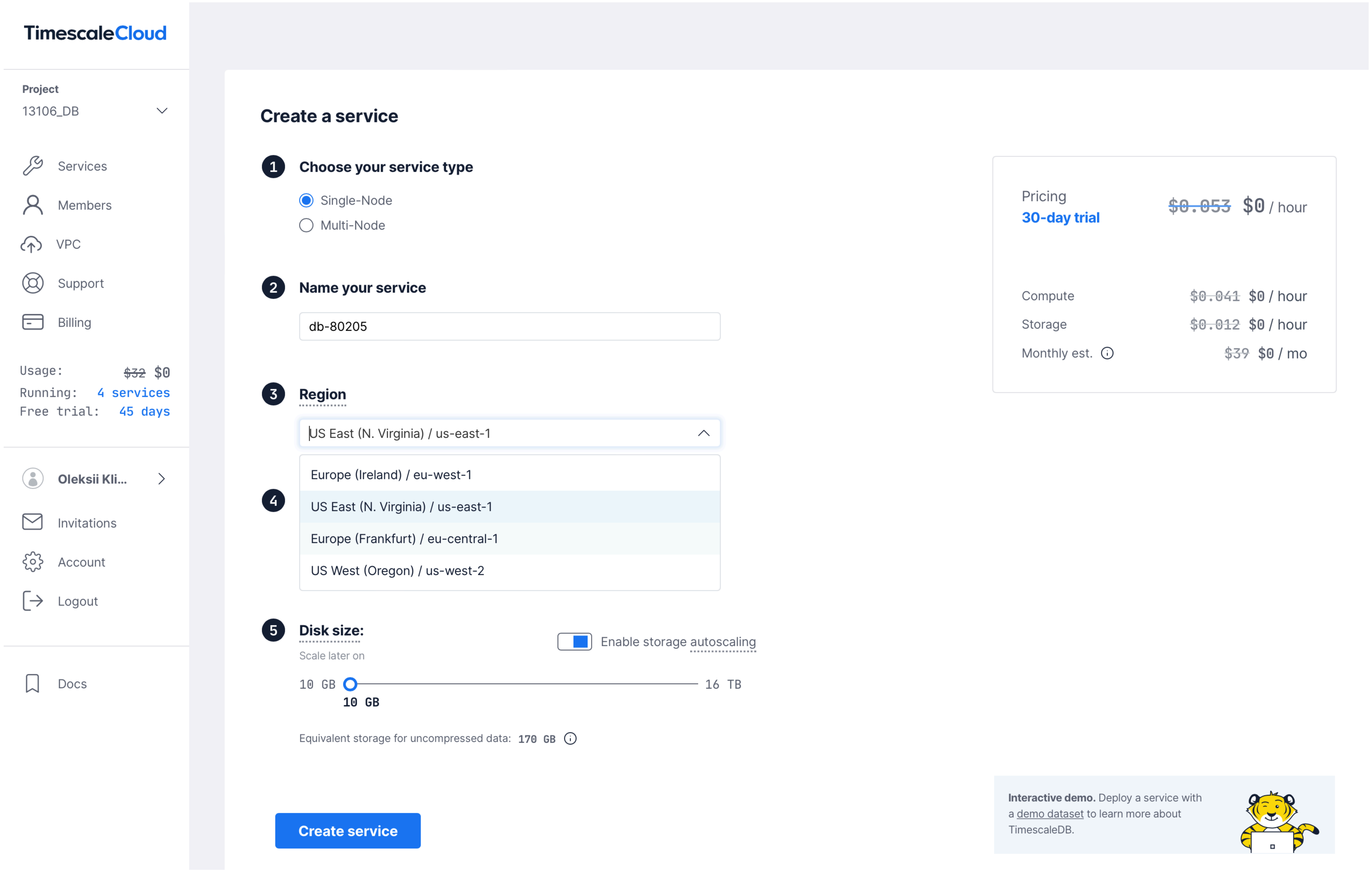
n9bhuqpzt1	1%	300 MB of 50 GB	s0rz61vu0e	0%	300 MB of 75 GB
h8ncp6ncv7	0%	300 MB of 75 GB	uyory1jwcp	0%	300 MB of 75 GB

Created 7 minutes ago

fork-db-13108 SINGLE-NODE Running
Forked from db-13108

CPU	RAM	Disk utilization	Region
0.5	2 GB	3% 350 MB of 10 GB	us-east-1

Created a day ago



TimescaleCloud

Project
13106_DB

```
Usage:       ~~$32~~ $0
Running:   4 services
Free trial:   45 days
```

Create a service

1 Choose your service type

[✓ Single-Node]
Multi-Node

2 Name your service

db-80205

3 Region

US East (N. Virginia) / us-east-1

- Europe (Ireland) / eu-west-1
- US East (N. Virginia) / us-east-1
- Europe (Frankfurt) / eu-central-1
- US West (Oregon) / us-west-2

4

5 Disk size:

Scale later on

Enable storage autoscaling [✓]

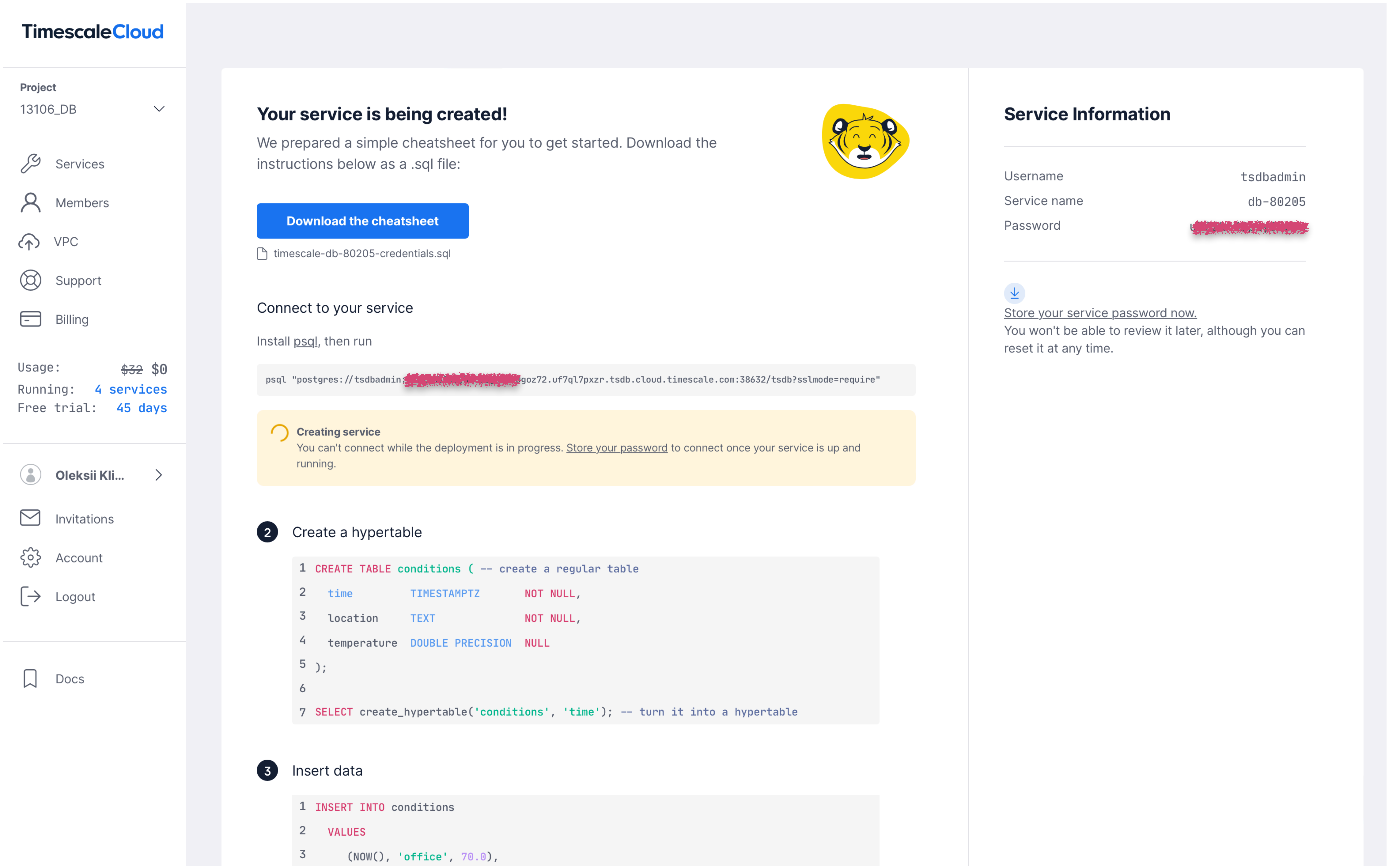
10 GB — 16 TB
10 GB

Equivalent storage for uncompressed data: 170 GB

Create service

Pricing 30-day trial	~~$0.053~~ $0 / hour
Compute	~~$0.041~~ $0 / hour
Storage	~~$0.012~~ $0 / hour
Monthly est.	~~$39~~ $0 / mo

Interactive demo. Deploy a service with a demo dataset to learn more about TimescaleDB.



TimescaleCloud

Project
13106_DB

- Services
- Members
- VPC
- Support
- Billing

```
Usage:        ~~$32~~ $0
Running:    4 services
Free trial:    45 days
```

- Oleksii Kli...
- Invitations
- Account
- Logout
- Docs

Your service is being created!

We prepared a simple cheatsheet for you to get started. Download the instructions below as a .sql file:

Download the cheatsheet

timescale-db-80205-credentials.sql

Connect to your service

Install psql, then run

```
psql "postgres://tsdbadmin:[redacted]goz72.uf7ql7pxzr.tsdb.cloud.timescale.com:38632/tsdb?sslmode=require"
```

Creating service
You can't connect while the deployment is in progress. Store your password to connect once your service is up and running.

2 Create a hypertable

```
CREATE TABLE conditions ( -- create a regular table
  time          TIMESTAMPTZ       NOT NULL,
  location      TEXT              NOT NULL,
  temperature   DOUBLE PRECISION  NULL
);

SELECT create_hypertable('conditions', 'time'); -- turn it into a hypertable
```

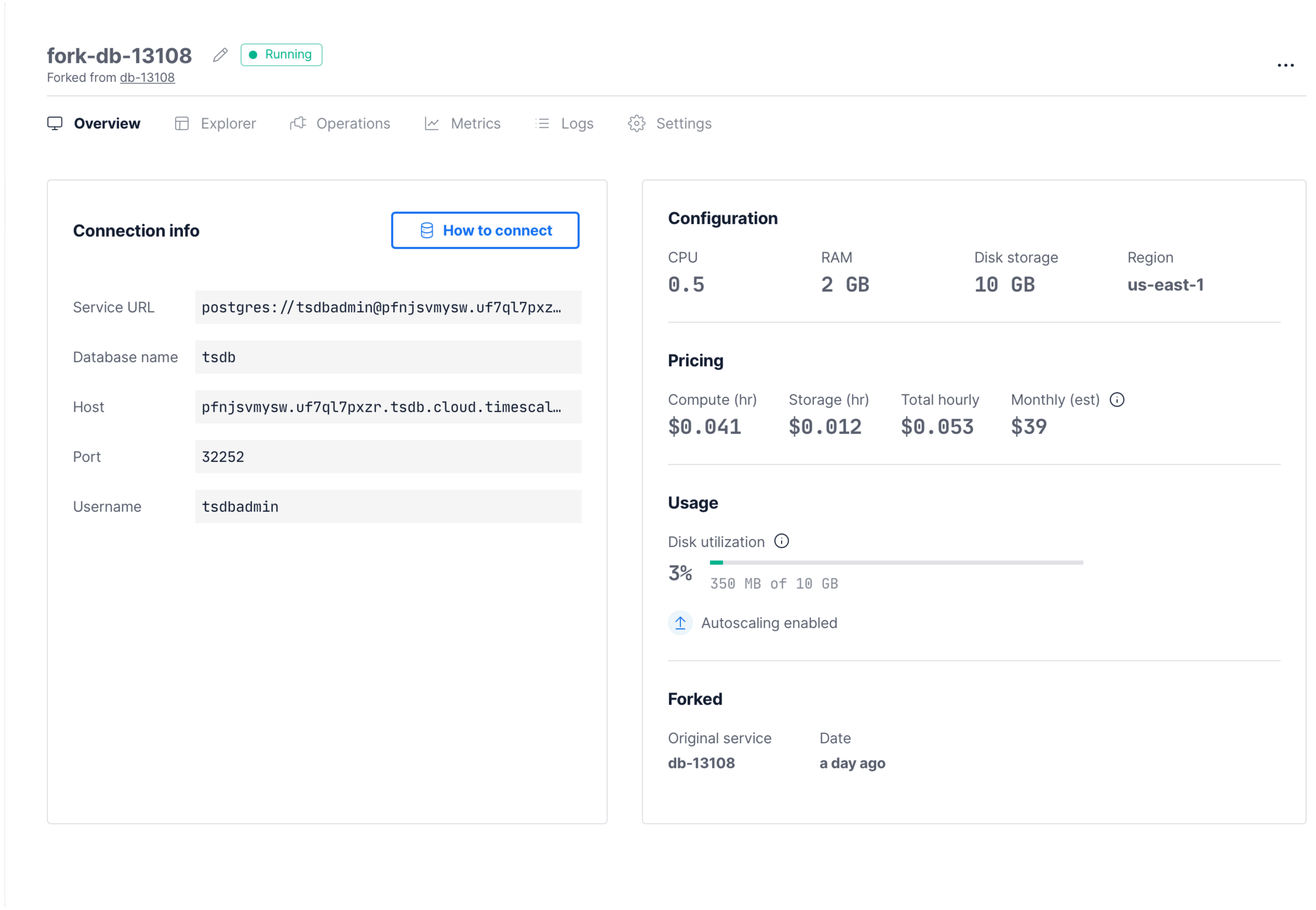
3 Insert data

```
INSERT INTO conditions
  VALUES
    (NOW(), 'office', 70.0),
```

Service Information

Username	tsdbadmin
Service name	db-80205
Password	[redacted]

Store your service password now.
You won't be able to review it later, although you can reset it at any time.


fork-db-13108 ● Running

Forked from db-13108

Overview | Explorer | Operations | Metrics | Logs | Settings

Connection info

How to connect

Service URL	postgres://tsdbadmin@pfnjsvmysw.uf7ql7pxz…
Database name	tsdb
Host	pfnjsvmysw.uf7ql7pxzr.tsdb.cloud.timescal…
Port	32252
Username	tsdbadmin

Configuration

CPU	RAM	Disk storage	Region
0.5	2 GB	10 GB	us-east-1

Pricing

Compute (hr)	Storage (hr)	Total hourly	Monthly (est)
$0.041	$0.012	$0.053	$39

Usage

Disk utilization

3% 350 MB of 10 GB

Autoscaling enabled

Forked

Original service	Date
db-13108	a day ago

Cloud DB architecture

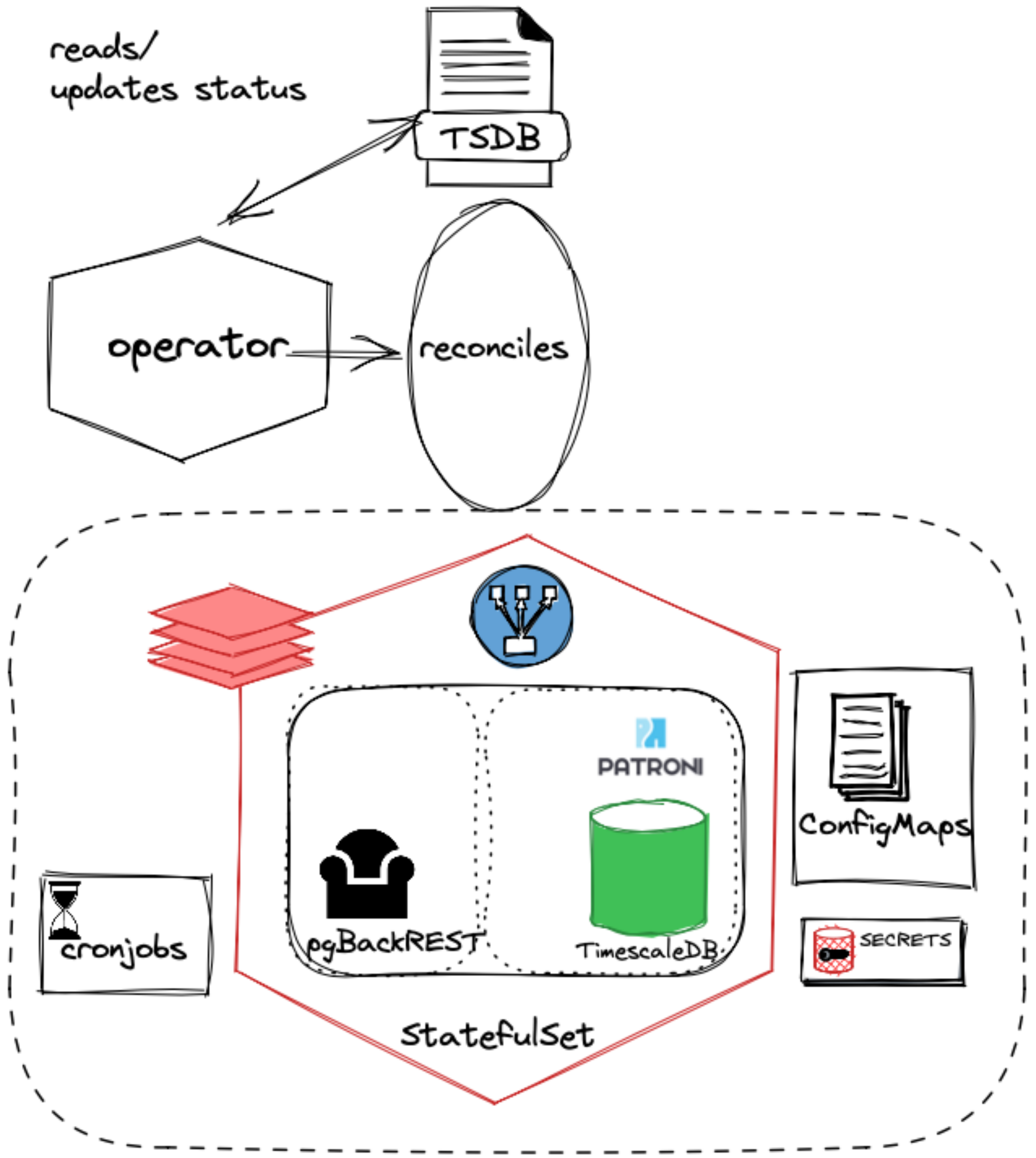
TimescaleDB CR object

```
$ kubectl get tsdb pgconfde

NAME          STATUS      AGE   NODES    VOLUME    CPU    MEMORY    BACKUP
pgconfde      Available   55d     an     20Gi      4      1Gi       true
```

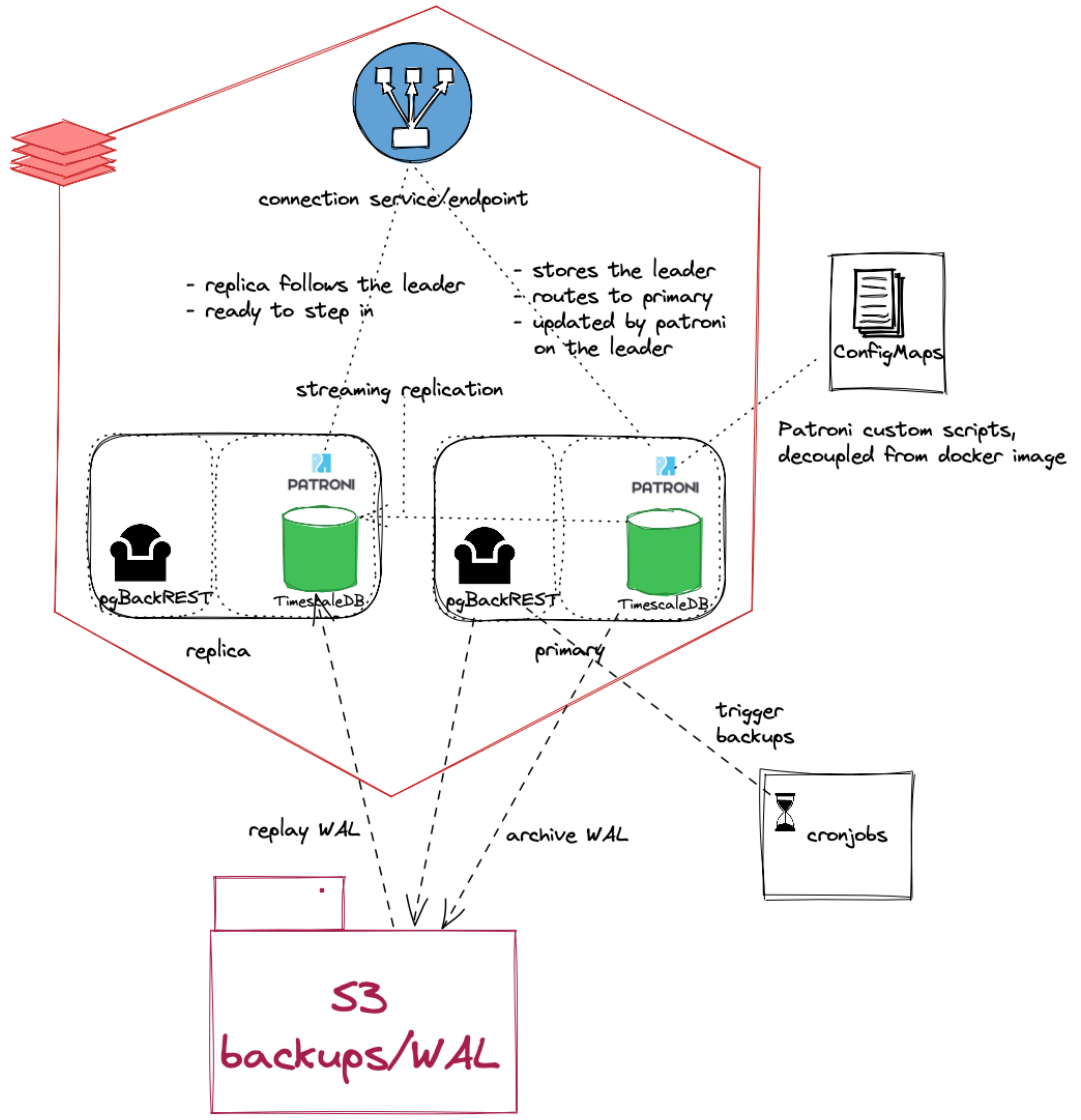
Cloud DB architecture

TimescaleDB Operator

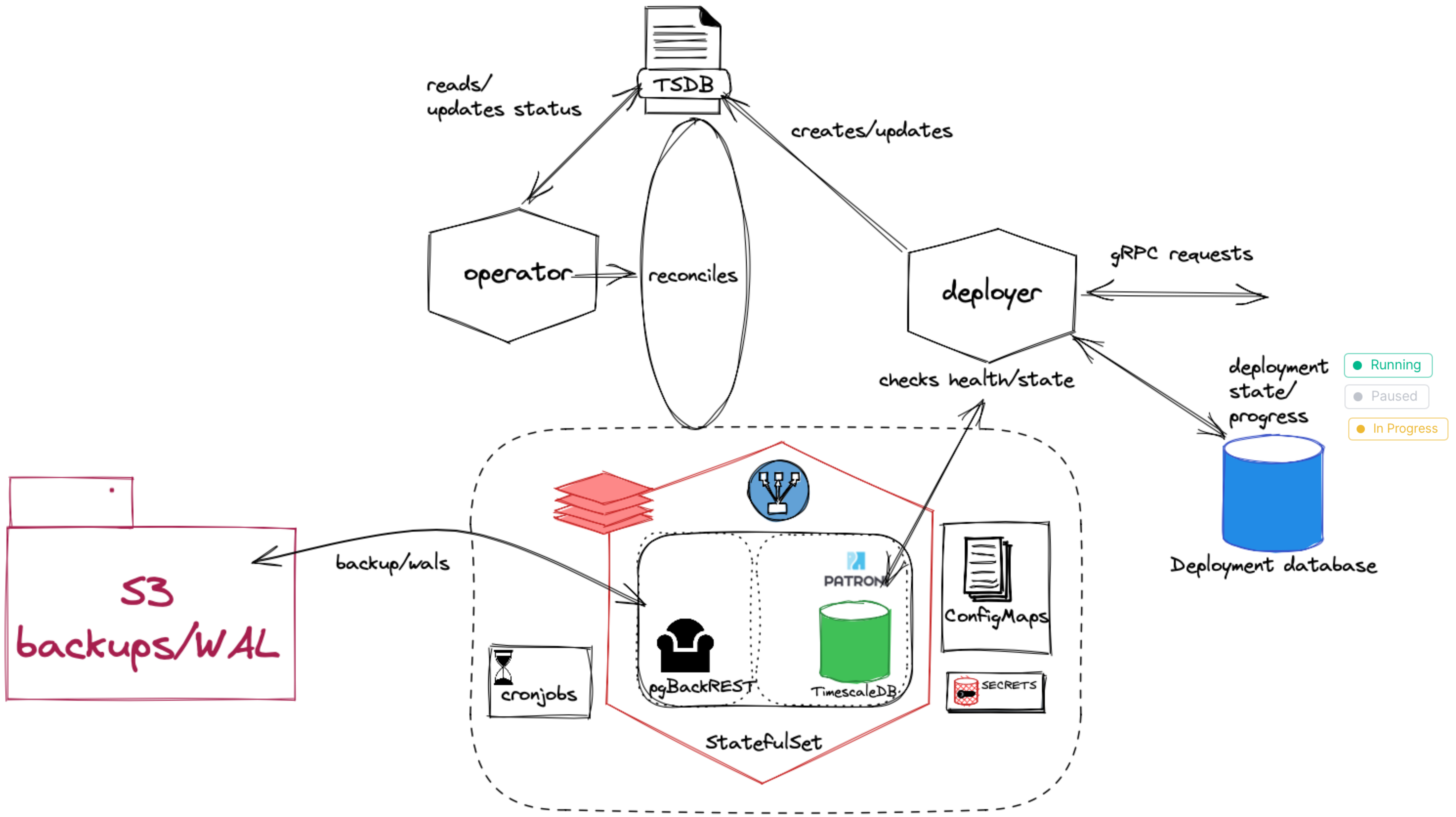


Cloud DB architecture

Inside pod



Timescale cloud DB on Kubernetes / AWS



Cloud DB architecture

Deployer - Operator split

- Operator reconciles Kubernetes objects, is essentially stateless.
- Deployer tracks the instance deployment events in a management database and determines whether the instance is ready by connecting to it and if necessary provisioning extensions, roles and permissions.
- Deployer writes TSDB spec, operator only reads the spec and updates the status.

Cloud DB architecture

Patroni managing Postgres container state

- Patroni is a template for Postgres HA written in Python
- Starts Postgres and keeps its state in a consistency layer (Postgres endpoint)
- Takes the leader lock if available, becoming a primary
- Initializes replicas from S3
- Restarts Postgres after the pod bounce
- Recovers from S3 when the volume is lost
- No dependency on microservices

Cloud DB architecture

Kubernetes advantages

- Automatic reproducible deployments
- Labels and annotations on Kubernetes objects for testing and safe production rollouts
- Informers and watches for availability checks and actions on every running instance
- Resources configuration to provide a wide variety of CPU/ memory combinations, not limited by VM granularity
- Auto-recovery from crashes

Cloud DB architecture

Service recovery after failure

- Pod failure: failover or restart by a StatefulSet
- Persistent volume failure: point in time recovery from S3
- Accidental TSDB deletion: restore definition from management database, point in time recovery from S3
- Complete loss of a Kubernetes cluster: restore management DB from S3, restore all TSDBs as if they were deleted

02

Challenges

And solutions

Challenges
OOM killer

OOM causes abrupt shutdown of PostgreSQL

- Timescale continuous/real-time aggregates may require a lot of memory. Out of memory (OOM) when limits are set low is not uncommon.
- OOM behavior assumed by PostgreSQL:

```
ERROR: out of memory on a request of 1024 bytes
```

- Linux OOM killer: SIGKILL a random Postgres process
- A backend process is killed: disruption, restart of every connection
- A postmaster is killed: unclean shutdown, in extreme cases to startup instance

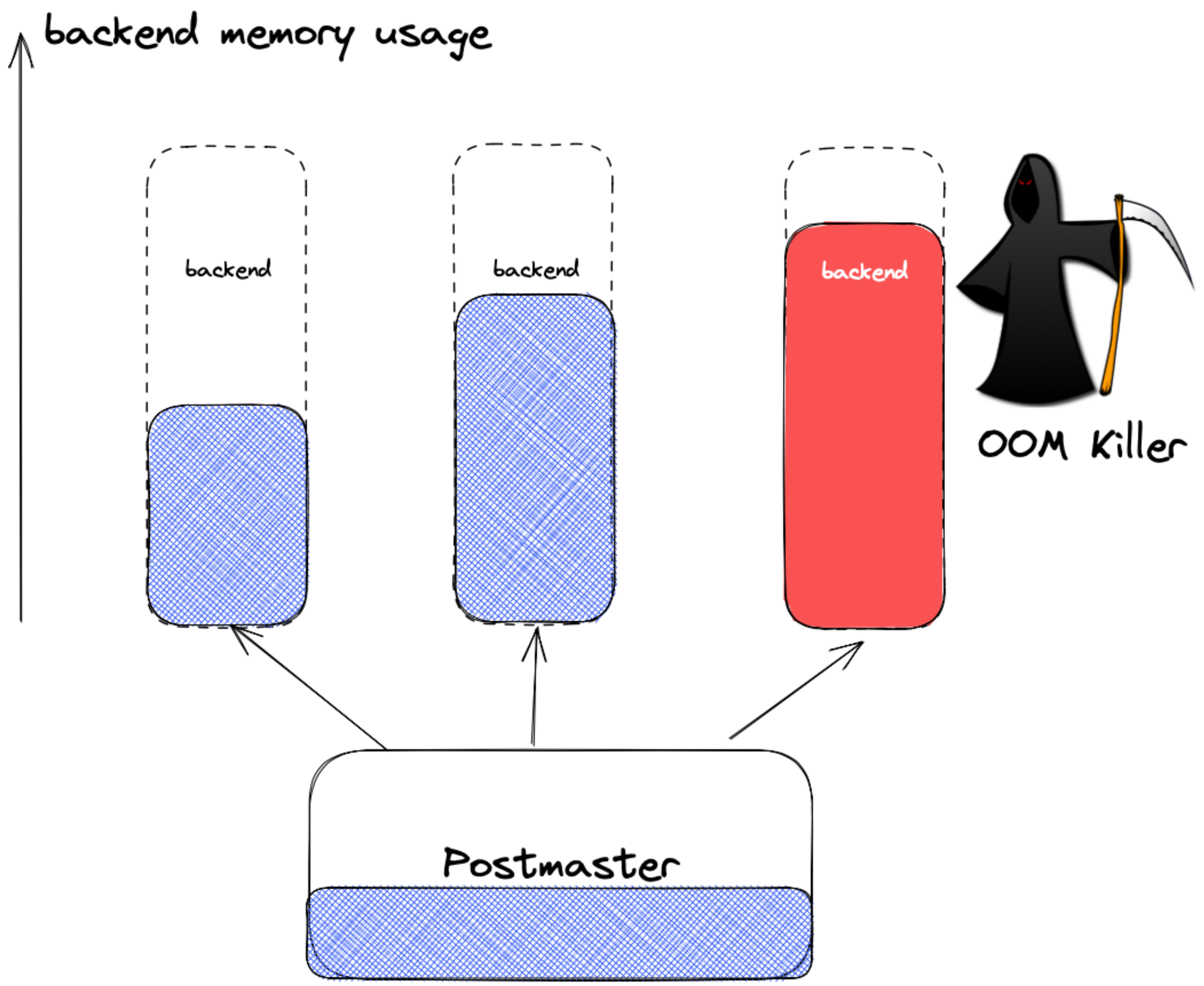
Challenges
OOM killer

OOM causes abrupt shutdown of PostgreSQL

```
$ kubectl get pod  tinyforkv01-an-0 -o json
jq '.spec.containers[0].resources'
{
  "limits": {
    "cpu": "4",
    "memory": "1Gi"
  },
  "requests": {
    "cpu": "4",
    "memory": "1Gi"
  }
}
```

Challenges
OOM killer

OOM causes abrupt shutdown of PostgreSQL



Challenges
OOM killer

OOM causes abrupt shutdown of PostgreSQL

- Regular PostgreSQL: set memory overcommit, enable swap
  - vm.overcommit_memory = 2
- Can’t set it individually per container
- A node typically runs some pods (eg. daemonsets for logging) incompatible with this setting

https://github.com/kubernetes/kubernetes/issues/90973

Challenges
OOM killer

OOM causes abrupt shutdown of PostgreSQL

- Solution: OOMGuard library collects statistics on the memory usage, overriding malloc
- Use LD_PRELOAD_LIBRARY to install it for Postgres processes
- Can just report statics, or actually block allocations going above the predefined threshold, emulating regular malloc behavior
- OOM_GUARD_LIMIT threshold is derived from the container memory limit, accounting for shared_buffers and OS overhead.

Challenges
OOM killer

Wishlist

- PostgreSQL: provide memory allocation hooks to do internal accounting and deny allocations via extensions.
- Linux/Kubernetes: configure oom_adj_score and vm_overcommit per cgroup on the Linux/Kubernetes layer.
- Improved debugging experience (locating debug symbols from the container when running perf or gdb on the host)

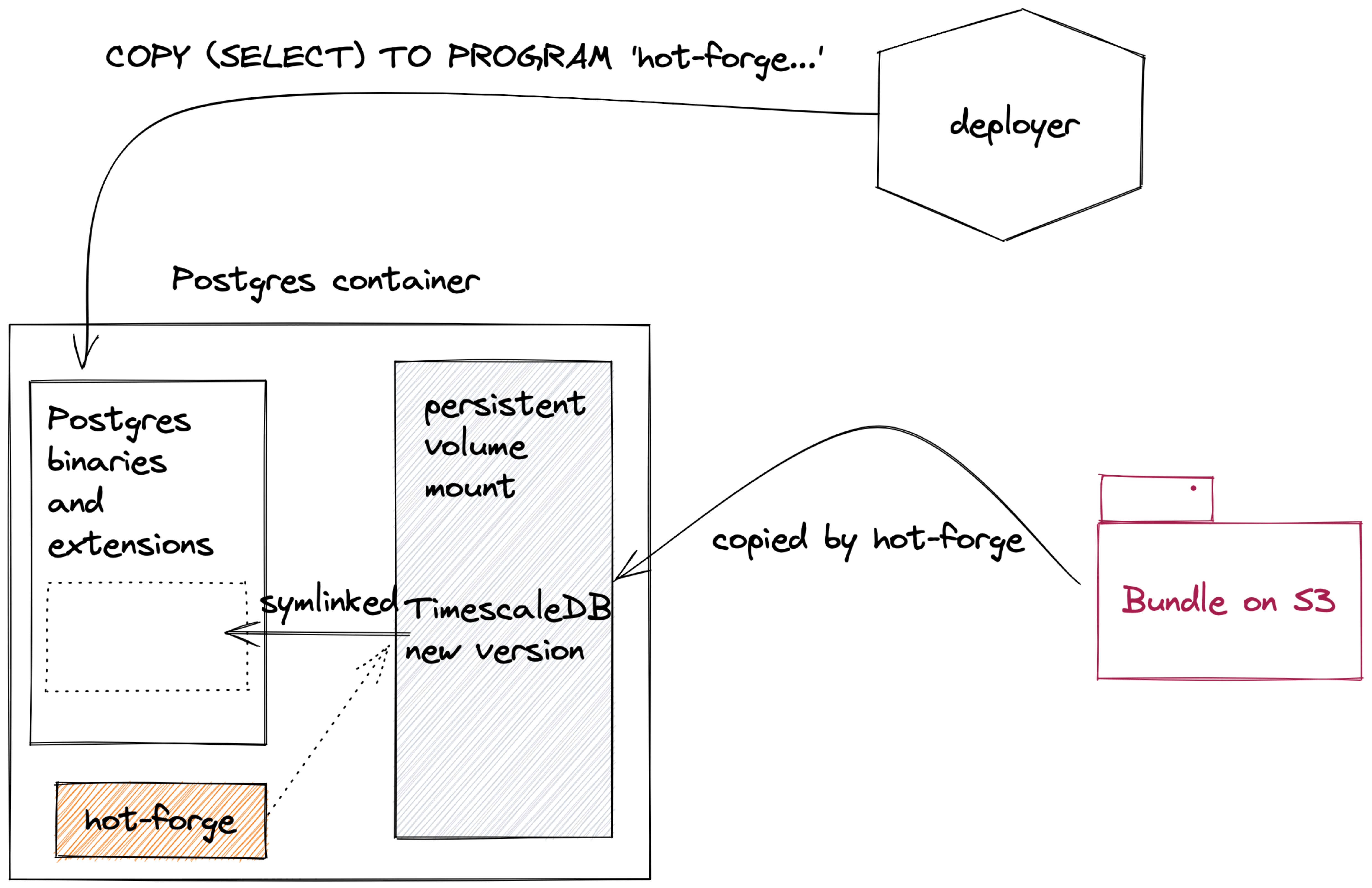
Challenges
Fewer downtimes

Extension updates require a pod bounce

- New versions of Timescale extension are released regularly
- A new timescale-docker-ha image is built once the extension is released
- We want to deliver latest extension (but not necessary auto-upgrade) to our customers immediately
- Changing pod’s docker image requires a pod restart
- Planned customer downtime may only happen during maintenance window, only a few times a year

Challenges
Fewer downtimes

Extension updates require a pod bounce



Challenges
Fewer downtimes

Extension updates require a pod bounce

- Solution: hot-forge
- A binary inside the container to fetch pre-packaged bundles and put them in the container
- The bundles are delivered using a postgres connection (COPY TO PROGRAM)
- The bundles are written to a persistent volume and linked to a container filesystem
- Mostly adding new data (although can potentially replace/delete existing files in the container)

Challenges
Fewer downtimes

Wishlist

- Allow bouncing of individual containers in the pod and changing the docker image
- Support “mutable” area inside the pod to deliver updates.

Challenges Operating Etcd

Etcd is a 5-nodes single point of failure

- Etcd is a core of the Kubernetes cluster
- Consists of multiple nodes (we run 5) - should be resilient?
- Can degrade on master node updates
- Performance issues (EBS burst balance, too many objects)
- Patroni dependency (no Kubernetes API - instances are read-only)

Challenges Operating Etcd

Etcd is a 5-nodes single point of failure

- Solution: no silver bullet
- Many small clusters in each region instead of a single big one
- Etcd performance monitoring
- Fire drills on ephemeral clusters
- Solution: Patroni experimental static_primary mode:
  - Enforce single primary by rejecting connections from other nodes
  - Do not demote when Kubernetes API is not available

Challenges Operating Etcd

Wishlist

- Some operational instructions when Etcd is down
- Better observability inside Etcd
- Patroni “isolated” mode scalable to any number of pods

Challenges
AWS bugs

Encrypted EBS volumes

- New (1TB+) encrypted EBS volumes show an existing partition marker (Atari partition)
- Kubernetes refuses to format them
- Pod is stuck at startup

Challenges
AWS bugs

Big encrypted EBS volumes

- Solution: create a small 1GB encrypted volume
- Snapshot it into a “golden snapshot”
- Create new encrypted volumes from the golden snapshot
- Need to resize the filesystem in the init container (as per Kubernetes 1.19)
- Recent fix by AWS: https://github.com/kubernetes/kubernetes/issues/86064

Challenges
AWS bugs

Big encrypted EBS volumes

```
$ kubectl get statefulset l0c154j810-an | jq
'.spec.volumeClaimTemplates[0].spec'
{
  "accessModes": [
    "ReadWriteOnce"
  ],
  "dataSource": {
    "apiGroup": "snapshot.storage.k8s.io",
    "kind": "VolumeSnapshot",
    "name": "golden-snapshot--wsboilqtlr"
  },
  "resources": {
    "requests": {
      "storage": "2500Gi"
    }
  },
  "storageClassName": "ebs-sc",
  "volumeMode": "Filesystem"
}
```

Challenges
AWS bugs

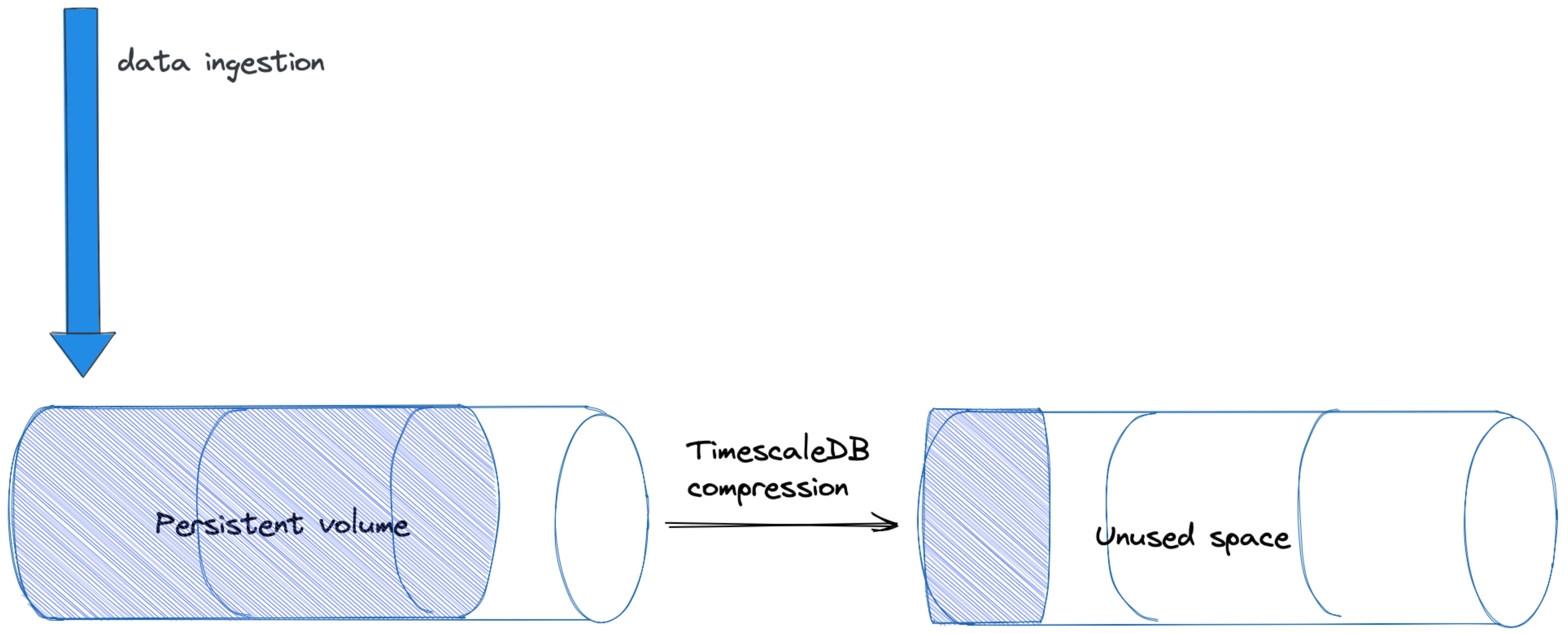
Wishlist

- Fewer bugs :-)
- Improved support for VolumeSnapshots, e.g. provisioning volumes across namespaces, resizing a filesystem when provisioning from a snapshot

Challenges
Volume resize

Volume size can only be increased, not decreased

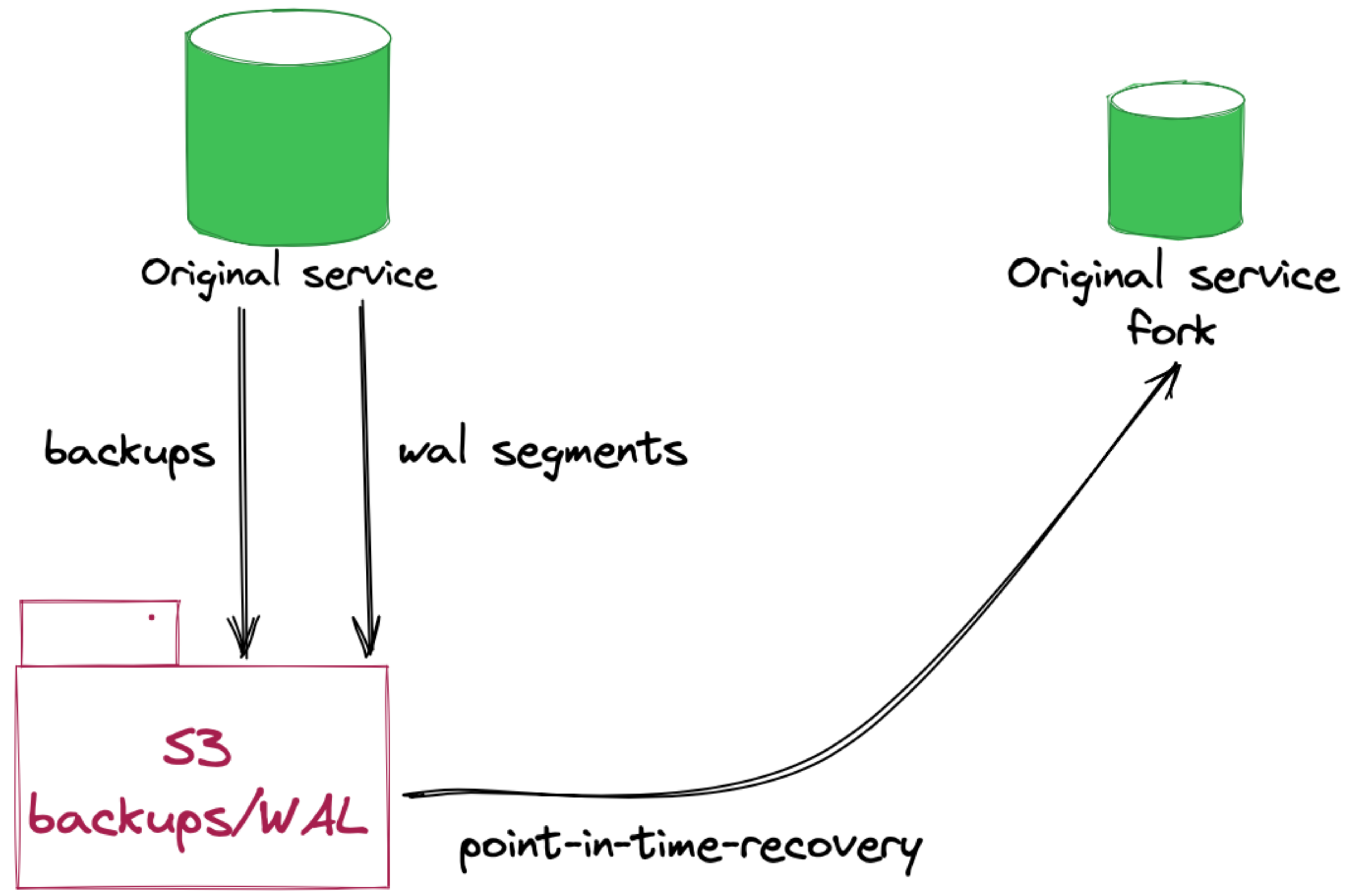
- AWS EBS and other PersistentVolume implementations only allow volume size increments.
- A volume autoscaler (Timescale service) may decide to increase the volume upon a data ingestion
- When data is subsequently compressed the customer doesn't need to pay for a bigger volume



Challenges
Volume resize

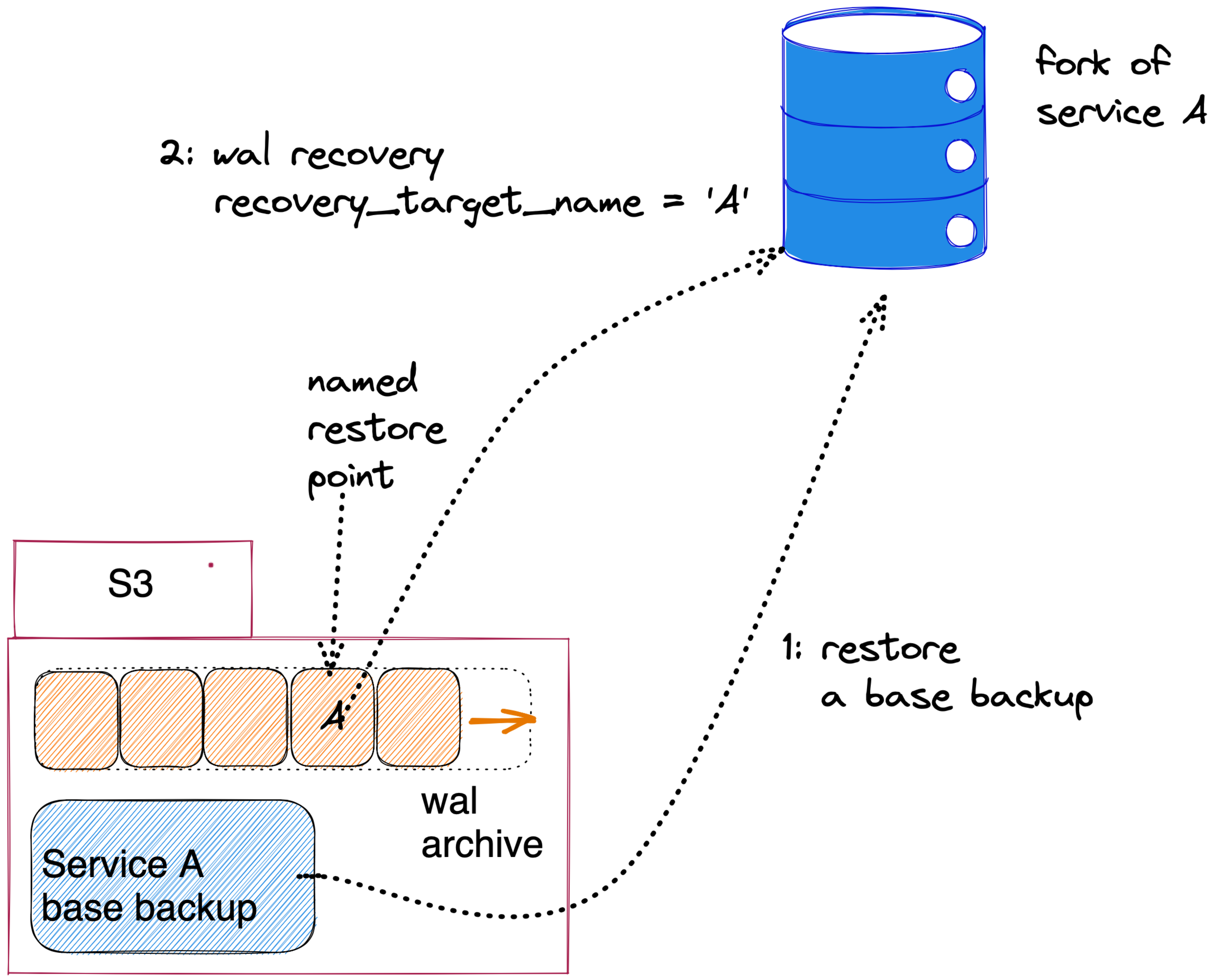
Volume size can only be increased, not decreased

- Solution: provide a functionality to fork a service
- A fork is a clone of a service with possibly different CPU and storage specs
- A fork is implemented by restoring another instance from the backup of the original one, taken from S3



Challenges
Volume resize

Forks zoom-in (Patroni custom boostrap)



Challenges
Volume resize

Wishlist

- Native volume downsize
- Kubernetes support, possibly with custom checks from K8s to determine this is possible.
- Support for volume resizing in a statefulset

Challenges
No superuser

Not giving out postgres superuser

- Can easily leak into a container (i.e. COPY TO PROGRAM)
- Need to provide an admin user to:
  - create other roles
  - create extensions
  - change some configuration parameters

Challenges
No superuser

Not giving out postgres superuser

- Admin user with CREATEROLE and CREATEDB
- CREATEROLE is too powerful:
  - Example: GRANT pg_execute_server_program TO adminuser
  - Use ProcessUtility hooks to stop unwanted grants
  - Allows "protecting" some roles from changes

```
tsdb=> GRANT pg_execute_server_program TO tsdbadmin
;

ERROR:  tsdb_admin: insufficient permission to
administer any default roles including
"pg_execute_server_program"
HINT:  Only superusers are allowed to administer
default roles
```

Challenges
No superuser

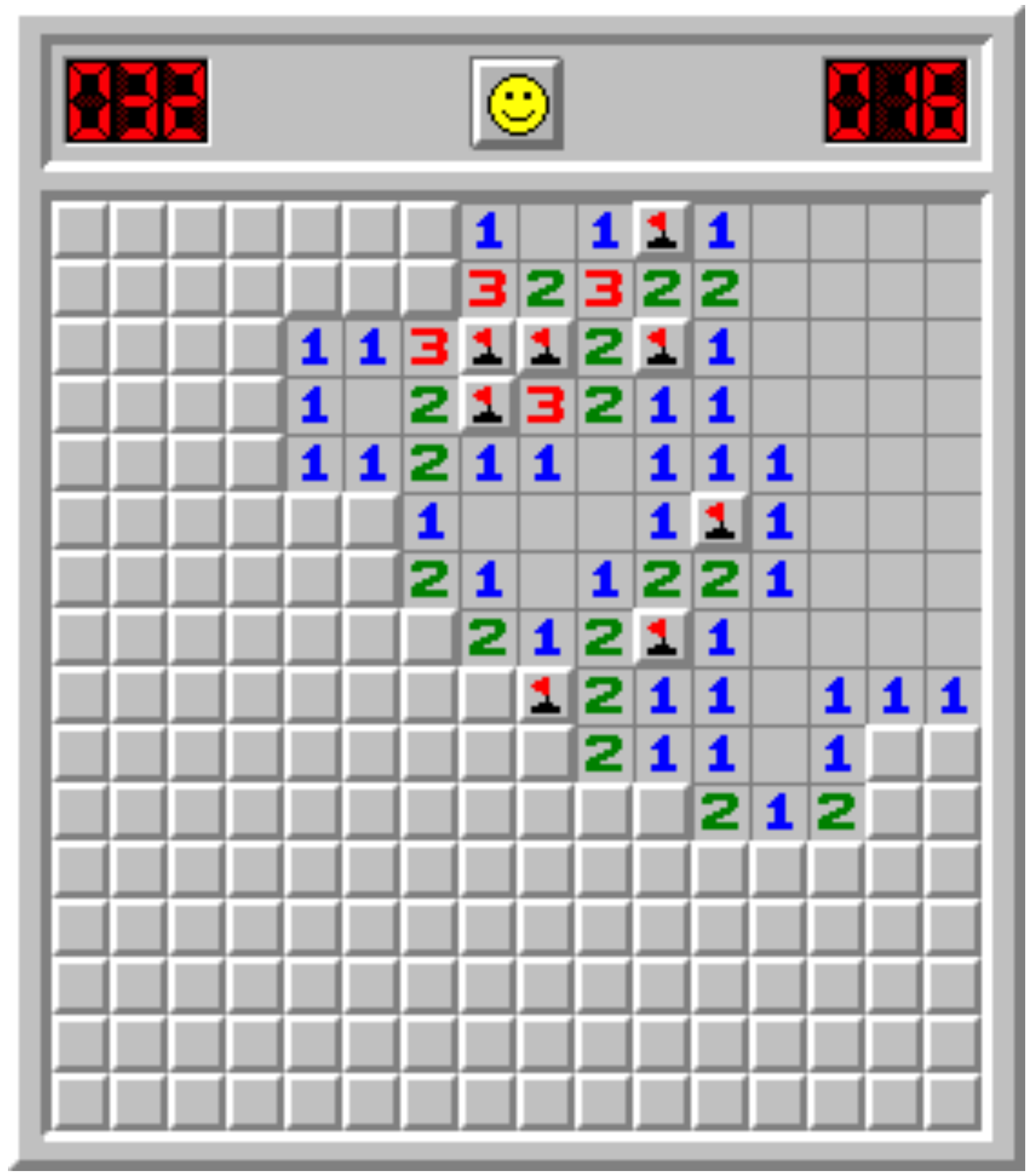
Installing extensions by non-superuser

- Whitelist extensions: https://github.com/dimitri/pgextwlist.git
- Similar to trusted extensions in v13
- Allows to list vetted extensions in guc
- Pre and post install-upgrade hooks to sanitize the DB
- Vulnerabilities checker: https://github.com/timescale/pgspot

Challenges
No superuser

Wishlist

- Deprecate superuser in PostgreSQL

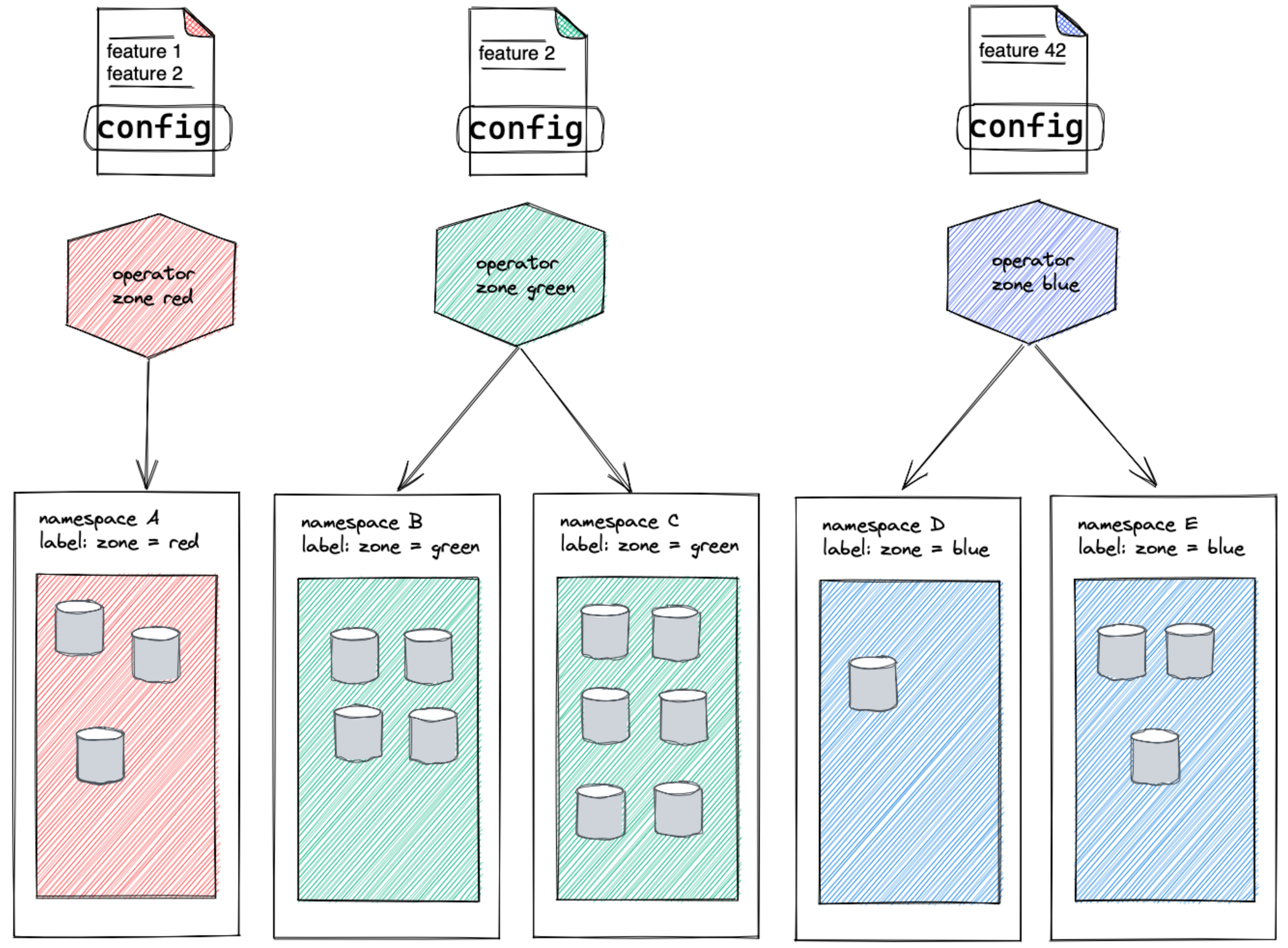


03

Developer experience

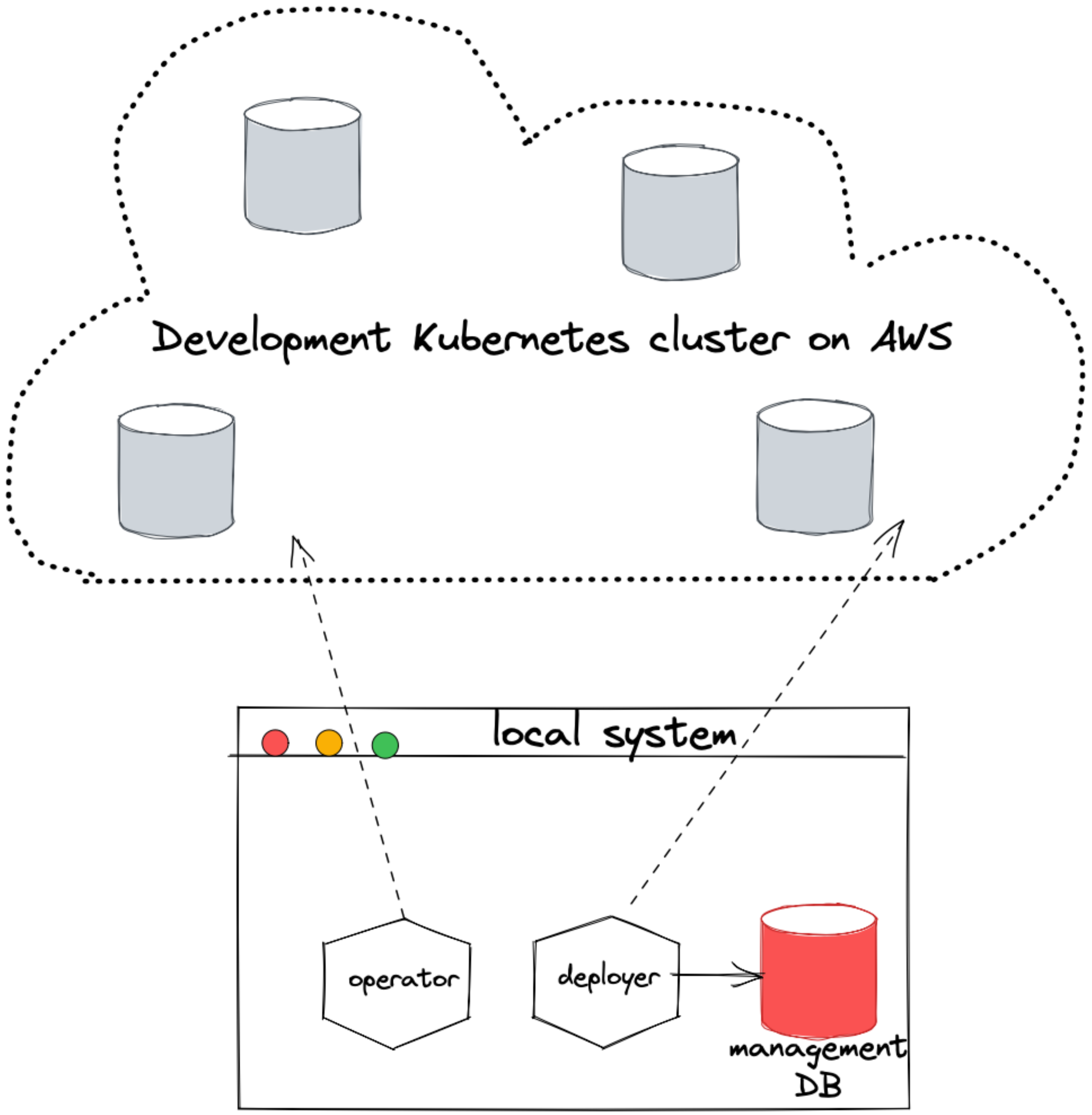
Developer experience

Feature flags and deployment zones


feature 1
feature 2
config
feature 2
config
feature 42
config
operator zone red
operator zone green
operator zone blue
namespace A label: zone = red
namespace B label: zone = green
namespace C label: zone = green
namespace D label: zone = blue
namespace E label: zone = blue

Developer experience

Out-of-cluster dev mode



Developer experience

Local development with Kind?

- Possible in principle
- Poor observability
- Additional burden of supporting running locally
- Not 1:1 environment
- Can’t test cloud-specific features (e.g EBS volume resize)

Developer experience

All the rest

- Deployer tests with actual database
- Operator tests in real Kubernetes environment
- Tests for the Docker image
- Dedicated dev environment
- CI/CD
- Can span new ephemeral Kubernetes cluster with only a couple of commands
- Tracing, centralized log collection, graphs and alerts
- Hands-off mode for the operator to disable reconcile for a TSDB instance

Questions

oleksii@timescale.com

Twitter: @hintbits

Thank you!

#AlwaysBeLaunching