Running Postgres-as-a-Service
In Kubernetes

@LukasFittl
Why (not) run Postgres in Kubernetes?
Consistency - Kubernetes manages all workloads, including databases
Better Portability - Consistent deployment experience across clouds, instead of relying on cloud-specific APIs
Low Latency - Co-locating Compute and Database, allows certain workloads to perform better
High Effort - Running anything in Kubernetes is complex, and databases are worse
How to deploy Postgres in Kubernetes
Postgres
Postgres High Availability
Scale Out Capabilities
Backups
Connection Pooling
Monitoring
K8S Integration
Zalando Postgres Operator

Postgres Operator
Crunchy Data PostgreSQL Operator
Azure Arc enabled data services

Bring Azure data services to on-premises, multi-cloud, and edge environments

Run Azure Arc enabled data services anywhere

Azure Arc allows you to run Azure data services on-premises, at the edge, and in multi-cloud environments using Kubernetes on the infrastructure of your choice. Get the latest Azure innovations, elastic scale, and unified management for data workloads with or without direct cloud connection.

Azure SQL Managed Instance and Azure Database for PostgreSQL Hyperscale enabled by Azure Arc are available in preview, with more Azure data services to come.
K8S Integration

CLIs, Operators & API Servers

Namespace handling

Storage
Zalando Postgres Operator

Operator

postgresql CRD

Postgres
Zalando Postgres Operator

Crunchy Data PostgreSQL Operator

"pgo" CLI

Operator

postgresql CRD

API Server

Pgcluster CRD

Pgpolicy CRD

Pgreplica CRD

Pgtask CRD

Postgres
Zalando Postgres Operator

kind: postgresql
metadata:
  ...
spec:
  databases:
    foo: zalando
  numberOfInstances: 2
  postgresql:
    version: "12"
  preparedDatabases:
    bar: {}
  teamId: acid
users:
  foo_user: 
  zalando:
    - superuser
    - createdb
volume:
  size: 1Gi

Crunchy Data PostgreSQL Operator

kind: Pgcluster
metadata:
  ...
spec:
  databases:
    foo: zalando
  numberOfInstances: 2
  postgresql:
    version: "12"
  preparedDatabases:
    bar: {}
  teamId: acid
users:
  foo_user: 
  zalando:
    - superuser
    - createdb
volume:
  size: 1Gi

PostgreSQL Hyperscale - Azure Arc

kind: DatabaseService
metadata:
  ...
spec:
  docker:
  ...
engine:
  type: Postgres
  version: 12
monitoring:
  ...
scale:
  shards: 2
scheduling:
  default:
    resources:
      requests:
        memory: 256Mi
service:
  port: 5432
  type: NodePort
storage:
  volumeSize: 16i
Namespace handling

Crunchy Data' Postgres Operator **Namespace Modes:**

**dynamic:**
Operator can create, delete, update any namespaces and manage RBAC. Operator requires ClusterRole privilege.

**readonly:**
Namespaces need to be pre-created and RBAC pre-configured. Operator requires ClusterRole privilege.

**disabled:**
Deploy to single namespace, no ClusterRole privilege required.
Generally, expect Persistent Volume Claims (PVCs) to be utilized for the database storage.

Crunchy Data PostgreSQL Operator also supports table spaces, to utilize different storage types within the same database server (be careful when using it)
Postgres

- K8S Integration
- High Availability
- Backups
- Monitoring
- Connection Pooling
- Scaling Capabilities
Scenario 1: Automated Failover within a K8S cluster
Scenario 1: Automated Failover within a K8S cluster
Scenario 1: Automated Failover within a K8S cluster

K8S Cluster 1

Operator

PG1
Primary

PG2
Primary

Node Failure

Promote
Scenario 1: Automated Failover within a K8S cluster
Scenario 2: Disaster Recovery to another K8S cluster

K8S Cluster 1

- PG1: Primary
- PG2: Secondary
- Sync Rep

K8S Cluster 2

- PG3: Secondary
- Async Replication
- Operator

Operator
Scenario 2: Disaster Recovery to another K8S cluster

Large-Scale Data Center Failure
## High Availability

<table>
<thead>
<tr>
<th></th>
<th>HA within Same K8S Cluster</th>
<th>HA across K8S Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zalando Postgres Operator</td>
<td>Built-In</td>
<td>Manual</td>
</tr>
<tr>
<td>Crunchy Data PostgreSQL Operator</td>
<td>Built-In</td>
<td>Manual</td>
</tr>
<tr>
<td>PostgreSQL Hyperscale - Azure Arc</td>
<td>Built-In</td>
<td>Manual</td>
</tr>
</tbody>
</table>
Pod Anti-Affinity

label: failure-domain.beta.kubernetes.io/region=westus2

failure-domain.../zone=0

failure-domain.../zone=1

failure-domain.../zone=2
<table>
<thead>
<tr>
<th></th>
<th>Local Volume Backups</th>
<th>Point-in-time-Restore</th>
<th>Offsite Backups</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Zalando</strong> Postgres Operator</td>
<td>n/a</td>
<td>Built-in (wal-e)</td>
<td>Built-in (wal-e)</td>
</tr>
<tr>
<td><strong>Crunchy Data</strong> PostgreSQL Operator</td>
<td>Built-in (pgBackRest)</td>
<td>Built-in</td>
<td>Built-in (Amazon S3)</td>
</tr>
<tr>
<td><strong>PostgreSQL Hyperscale - Azure Arc</strong></td>
<td>Built-in</td>
<td>Built-in</td>
<td>Built-in (K8S Volume Mount)</td>
</tr>
</tbody>
</table>
Postgres

- K8S Integration
- High Availability
- Backups
- Monitoring
- Connection Pooling
- Scaling Capabilities
<table>
<thead>
<tr>
<th>Monitoring</th>
<th>Metrics</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Zalando</strong>&lt;br&gt;Postgres Operator</td>
<td>not built in</td>
<td>not built in</td>
</tr>
<tr>
<td><strong>Crunchy Data</strong>&lt;br&gt;PostgreSQL Operator</td>
<td>Grafana</td>
<td>Built-in pgbadger</td>
</tr>
<tr>
<td><strong>PostgreSQL Hyperscale</strong>&lt;br&gt;- Azure Arc</td>
<td>Grafana + Azure Monitor</td>
<td>Kibana + Azure Log Analytics</td>
</tr>
</tbody>
</table>
PostgreSQL Hyperscale - Azure Arc
pgbouncer is important for **idle connection scaling** in Postgres

Idle connection in Postgres: 5-10MB  
Idle connection in pgbouncer: <1MB
## Connection Pooling

<table>
<thead>
<tr>
<th></th>
<th>Pgbouncer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zalando Postgres Operator</td>
<td>Built-in</td>
</tr>
<tr>
<td>Crunchy Data PostgreSQL Operator</td>
<td>Built-in</td>
</tr>
<tr>
<td>PostgreSQL Hyperscale - Azure Arc</td>
<td>Planned</td>
</tr>
</tbody>
</table>
Postgres

- K8S Integration
- High Availability
- Backups
- Monitoring
- Connection Pooling
- Scaling Capabilities
Read Replicas:

Help you scale the read performance; Max data size = max storage size per node
Scaling Capabilities

Hyperscale (Citus):

Scales both read and write performance; Max data size = # of data nodes * storage size per node
Scaling Up

Block growth on 1 (monolithic) database

Scaling Out

Grow to 100's of database nodes, without re-architecting your application
Demo

Deploying & Scaling out with 
PostgreSQL Hyperscale - Azure Arc
Thank you!

lukas@fittl.com

@LukasFittl