Postgres Extensions in Kubernetes

Alvaro Hernandez
@ahachete

KubeCon | CloudNativeCon

North America 2021

`whoami`

- Founder & CEO, OnGres
- 20+ years Postgres user and DBA
- Mostly doing R&D to create new, innovative software on Postgres
- Frequent speaker (100+ talks)
- Principal Architect of StackGres, ToroDB
- Founder of the NPO Fundación PostgreSQL
- AWS Data Hero

aht.es

Postgres Extensions 101

Alvaro Hernandez *@ahachete*

What are Postgres extensions?

- One of the most important Postgres features. **Extensibility**.
- Postgres extensions are like browser plugins: bundles that augment Postgres functionality.
- Extensions allow out-of-core feature development:
  - Developed by third parties.
  - Not tied to Postgres development lifecycle.
- In many cases, they prevent Postgres hard forks.
- Most of the existing extensions are open source.

Postgres extensions examples

PostGIS: very advanced GIS database capabilities

CitusData: sharding, distributed queries
pg_auto_failover: HA as a Postgres extension

Timescale: time-series data on Postgres

ZomboDB: index that is a remote ElasticSearch

What can extensions do?

- Add new database objects like:
  - data types
  - functions, including aggregates, operators
  - procedural languages.
- Table and index access methods (“storage engines”).
- Insert/replace extension hooks.
  Call any Postgres internal method.
- Modify WAL, replication, create background workers...
- Package additional binaries and shared libraries.

What languages are supported?

- SQL, plpgsql and any procedural language: database objects.
- C: for anything, including database objects and calling internal APIs. Most extensions are probably C extensions.
- In general, any language that can compile to a shared lib:
  - Rust! E.g. ZomboDB
  - C++
  - ... (waiting for the next innovations!)

How to use an extension

- Installed in: `/usr/share/postgresql/{majorVersion}/extension` `.control`, `.sql` and other files.
- Once installed:
  - C extensions compiled as shared libraries may need to be previously loaded into shared_preload_libraries: `shared_preload_libraries='extension1,extension2,...'` This requires a database restart.
  - `CREATE EXTENSION extension_name; --per database`
  - Extensions may add their own GUCs in `postgresql.conf`

Installing extensions

- Where to find them? **There's no "extension store"!** DuckDuckGo or Google them.
- Some are packaged in debs/rpms/etc.
- pgxn is a repository for many. But only in source code form!
- Most of the time, you need to compile them:
  - Download/clone source code.
  - Have development dependencies. May require extension's own development dependencies.

Postgres Extensions in K8s: the Problems

Alvaro Hernandez *@ahachete*

Limited set of extensions in Cloud Postgres-aaS

Postgres 13 core distribution includes 76 extensions:
8 PL extensions + 68 contrib extensions

	Extensions	Core avail.	Core n/a	Third-party
AWS RDS	**73**	39	27	34
AWS Aurora	**76**	38	28	38
Cloud SQL	**52**	34	32	18
Azure Flexible*	**63**	44	22	19
Azure Hyper.	**63**	48	18	15

Also limited in Postgres Operators ?

Postgres 13 core distribution includes 76 extensions:
8 PL extensions + 68 contrib extensions

	Extensions	Core avail.	Core n/a	Third-party
Zalando 1.7.0	**89**	50	16	39
Crunchy 5.0.1	**58**	52	14	6
Kubegres 1.9	**44**	22	22	0
StackGres 1.0.0	**100+, dynamic***			

“Fat” Container problems

Add all the extensions to the container image.
Problems:

- Size of the resulting image?
- Update image every time add / remove / update extension.
- Restarts required!

https://www.flickr.com/photos/gavinbell/535261899

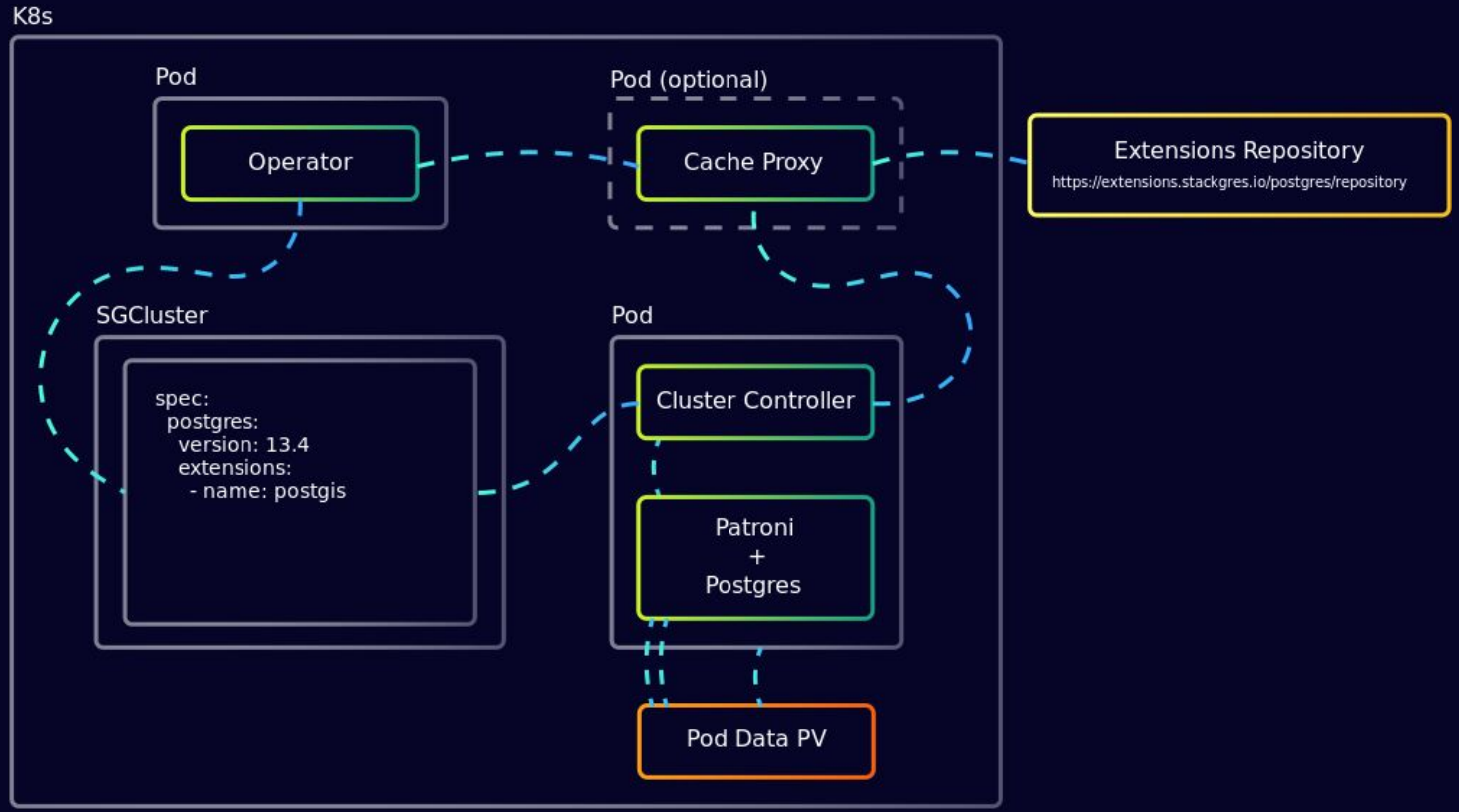
StackGres: introducing dynamic Postgres Extension loading

Alvaro Hernandez *@ahachete*

Dynamic Postgres extension loading

- No extension is built into the container image.
- Pod's ephemeral filesystem is not used for extensions. **Extensions are downloaded automatically by StackGres into the pod's PV**.
- **Postgres binaries are "relocated" also to the PV**, extensions symlinked. Also useful for major version upgrades!
- **A "cluster-controller" (a controller sidecar) runs a reconciliation cycle to load/unload extensions**, do symlinks.
- Extensions are pulled from a remote extension repository.

Dynamic Postgres extensions: Architecture



Dynamic Postgres extensions: how it works

```yaml
apiVersion: stackgres.io/v1
kind: SGCluster
metadata:
  name: kubecon
  namespace: cnc
spec:
  postgres:
    version: latest
  instances: 2
  pods:
    persistentVolume:
      size: '10Gi'
```

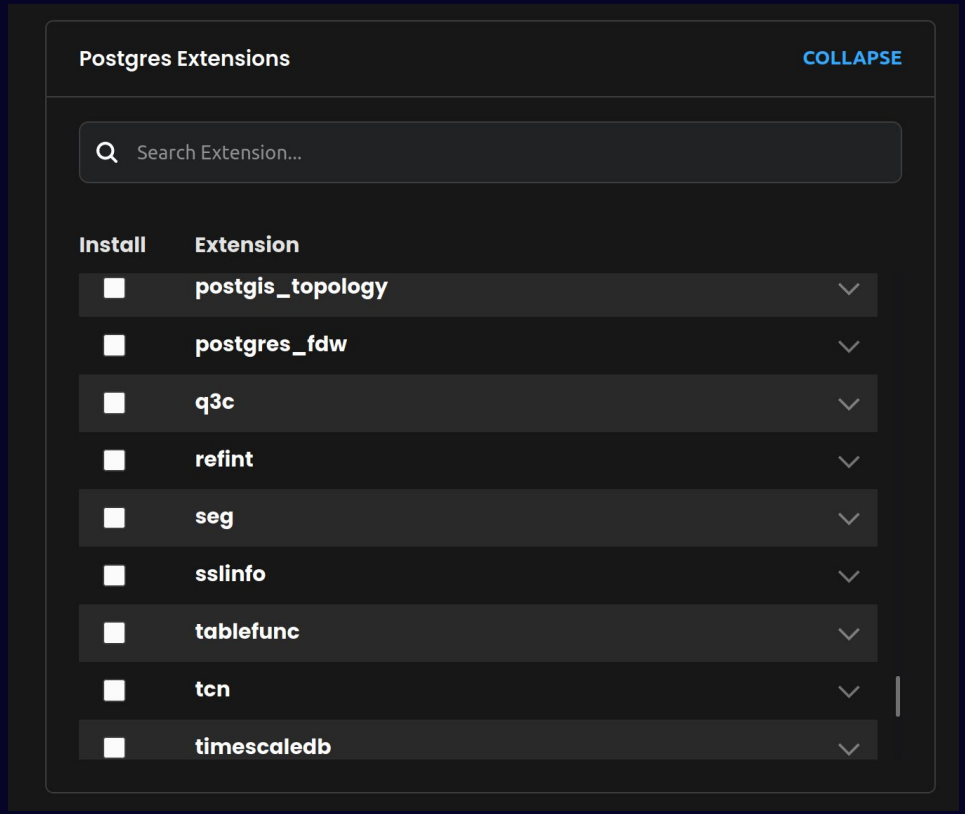
Create a basic SGCluster CR.

Dynamic Postgres extensions: how it works

```yaml
apiVersion: stackgres.io/v1
kind: SGCluster
metadata:
  name: kubecon
  namespace: cnc
spec:
  postgres:
    version: latest
    extensions:
      - name: postgis
  instances: 2
  pods:
    persistentVolume:
      size: '10Gi'
```

Add an extension (extension version is optional).

Which extensions are available?



Either via the Web Console (which provides search functionality and available versions) or the documentation.

(Or the hard way, jq-ing the repository)

Dynamic Postgres extensions: how it works

```
spec:
  postgres:
    version: "13.4"
    extensions:
    - name: postgis
  toInstallPostgresExtensions:
  - build: "6.4"
    extraMounts:
    - /usr/share/proj
    name: postgis
    postgresVersion: "13"
    publisher: com.ongres
    repository: https://extensions.stackgres.io/postgres/repository
    version: 3.0.1
```

Mutating webhook retrieves extension repository metadata, and details extension information in the toInstall section. Emits a warning if extension is not found.

Dynamic Postgres extensions: how it works

```
spec:
  postgres:
    version: "13.4"
    extensions:
    - name: postgis
  toInstallPostgresExtensions:
  - build: "6.4"
    extraMounts:
    - /usr/share/proj
    name: postgis
    postgresVersion: "13"
    publisher: com.ongres
    repository: https://extensions.stackgres.io/postgres/repository
    version: 3.0.1
```

The validating webhook verifies that there is a matching between the requested extension and the toInstall section (i.e., it was found). Otherwise, process is rejected.

Dynamic Postgres extensions: how it works

```
spec:
  toInstallPostgresExtensions:
  - build: "6.4"
    extraMounts: [ /usr/share/proj ]
    name: postgis
    postgresVersion: "13"
    publisher: com.ongres
    repository: https://extensions.stackgres.io/postgres/repository
    version: 3.0.1
status:
  podStatuses:
  - name: kubecon-0
    installedPostgresExtensions:
    - build: "6.4"
      extraMounts: [ /usr/share/proj ]
      name: postgis
      postgresVersion: "13"
      publisher: com.ongres
      repository: https://extensions.stackgres.io/postgres/repository
      version: 3.0.1
  - name: kubecon-1
    installedPostgresExtensions:
      ...
```

The **cluster controller**, running on each pod, installs the extensions (if needed) in the PV.

It then updates the respective `.status` field of the SGCluster.

Dynamic Postgres extensions: how it works

```
status:
  conditions:
  - lastTransitionTime: "2021-09-13T13:58:47.799446Z"
    reason: FalseFailed
    status: "False"
    type: Failed
  - lastTransitionTime: "2021-09-13T14:02:05.831007Z"
    reason: PodRequiresRestart
    status: "True"
    type: PendingRestart
```

The cluster controllers also inform the `.status` on whether anything failed (signaling the condition) and if the cluster requires a restart, to reload shared libraries imported by the extension.

Demo!!

Summary: Postgres Extensions in StackGres

- Postgres container is **extension-less**, no extensions included:
  - Increase security.
  - Decrease container size.
- You can load/unload/upgrade dynamically extensions, **in a declarative way**: adding them to the SGCluster CR.
- StackGres introduced a “cluster-controller”, a controller that runs as a sidecar in the Postgres pod. It downloads, unpacks, verifies and install extensions.
- StackGres comes with 100+ extensions from an external repository. **It will continue growing to multiple hundreds!**

Questions?

Join Community in Slack/Discord for following-up after the talk!

slack.stackgres.io

discord.stackgres.io