# Esta Tekton Results - Setup and Evaluation

## Content

* [Content](#EstaTektonResultsSetupandEvaluation-Con)
* [Introduction](#EstaTektonResultsSetupandEvaluation-Int)
  + - [Result](#EstaTektonResultsSetupandEvaluation-Res)
    - [Record](#EstaTektonResultsSetupandEvaluation-Rec)
  + [Tekton-Results instances](#EstaTektonResultsSetupandEvaluation-Tek)
* [Setup](#EstaTektonResultsSetupandEvaluation-Set)
* [Setup Namespaced Tekton Results](#EstaTektonResultsSetupandEvaluation-Set)
* [Some usefull SQL commands](#EstaTektonResultsSetupandEvaluation-Som)
  + [Test with Index creations](#EstaTektonResultsSetupandEvaluation-Tes)
  + [Delete Schema and User](#EstaTektonResultsSetupandEvaluation-Del)
  + [Reset Password of user](#EstaTektonResultsSetupandEvaluation-Res)
* [Tekton Result Setup Strategies](#EstaTektonResultsSetupandEvaluation-Tek)
  + [Global Results](#EstaTektonResultsSetupandEvaluation-Glo)
  + [Namespace Scoped Watcher](#EstaTektonResultsSetupandEvaluation-Nam)
* [DB Setup Options Tekton Result - Namespaced](#EstaTektonResultsSetupandEvaluation-DBS)
  + [Option 1 Standard - Postgresql Setup per Namespace by Deployment](#EstaTektonResultsSetupandEvaluation-Opt)
  + [Option 2 PGO Operator - Postgresql Setup per Namespace by Custom Resource (schematic picture same as above)](#EstaTektonResultsSetupandEvaluation-Opt)
  + [Option 3 Single Cluster DB Instance - Self managed Postgresql Instance on Aws01t](#EstaTektonResultsSetupandEvaluation-Opt)
  + [Option 4 Sigle RDS Instance - Multiple Schemas in one DB](#EstaTektonResultsSetupandEvaluation-Opt)
* [Setup Esta-Tekton RDS Schemas](#EstaTektonResultsSetupandEvaluation-Set)

## Introduction

To store the Tekton PipelineRun and TaskRun informations we will use either our own solution or a standard solution like [Tekton-Results](https://github.com/tektoncd/results).

Tekton Results watches all the Tekton Resources for changes and stores them into it's Postgresql DB. Tekton-Results DB fills two tables.

##### Result

In the Result table all parent objects and its relations to its child objects are stored. In our case PipelineRuns with the relations to its TaskRuns and PipelineRuns which are stored as YAML in the records table.

##### Record

In the record table all PipelineRun and TaskRun Yamls are stored.

Tekton-Results consist of 2 main components: the api and the watcher service. The Watcher service listens to all Tekton resource changes and reads the resources, stores them over the api service into the DB. The api service offers read and write access through GRPC or REST Api.

### Tekton-Results instances

We built up 2 Tekton-Results instances with following details:

|  |  |  |
| --- | --- | --- |
|  | **DEV - Clew01d** | **PRD - Aws01t** |
| Openshift Namespace | <https://console-openshift-console.apps.clew01d.sbb-aws-dev.net/k8s/cluster/projects/esta-tekton-results> | <https://console-openshift-console.apps.aws01t.sbb-aws-test.net/project-details/ns/esta-tekton-results> |
| PostgreSQL Db | <https://ssp.dbms.sbb.ch/manageinstanceaws?i=esta-tekton-dev>  The credentials you can find in the Keypass file. | <https://ssp.dbms.sbb.ch/manageinstanceaws?i=esta-tekton-prod> |
| S3 Bucket for the PipelineRun/TaskRun Logs  Config happens in  **tekton-results-api-config** | S3\_BUCKET\_NAME=esta-tekton-results-dev  S3\_ENDPOINT=https://s3.eu-central-1.amazonaws.com  S3\_HOSTNAME\_IMMUTABLE=false  S3\_REGION=eu-central-1  S3\_ACCESS\_KEY\_ID=<Keypass>  S3\_SECRET\_ACCESS\_KEY=<Keypass> | Not yet setup |
| The Watcher is mainly configured through the start parameter of the docker image which you can see in this piece of sourcecode:  <https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-results/browse/charts/esta-tekton-results/templates/deployment.yaml#94> | args:                       - -api\_addr                       - tekton-results-api-service.{{ .Values.namespace }}.svc.cluster.local:50051                       - -completed\_run\_grace\_period                       - '120h'                       - -namespace                       - esta-tekton-predev                       - -auth\_mode                       - insecure |  |
| REST Urls | https://tekton-results-api-service-tekton-pipelines.apps.clew01d.sbb-aws-dev.net/apis/results.tekton.dev/v1alpha2/parents/esta-tekton-predev/results | https://rest-esta-tekton-results.apps.aws01t.sbb-aws-test.net/apis/results.tekton.dev/v1alpha2/parents/-/results |
| Argocd Repo | <https://code.sbb.ch/projects/KD_PAAS/repos/ocp-argocd/browse/projects/esta/applications/tekton-results>  <https://code.sbb.ch/projects/KD_PAAS/repos/ocp-argocd/browse/projects/esta/values/groups/stage/dev/tekton-results.yaml> | <https://code.sbb.ch/projects/KD_PAAS/repos/ocp-argocd/browse/projects/esta/applications/tekton-results>  <https://code.sbb.ch/projects/KD_PAAS/repos/ocp-argocd/browse/projects/esta/values/groups/stage/test/tekton-results.yaml> |
| ArgoCD's | <https://ocp-argocd.sbb-cloud-dev.net/applications?showFavorites=false&proj=&sync=&health=&namespace=&cluster=&labels=> | <https://ocp-argocd.int.sbb-cloud.net/applications?showFavorites=false&proj=&sync=&health=&namespace=&cluster=&labels=> |

## Setup

The setup happens mainly through the ocp-argcd repo and Argocd instances. Afterwards some modifications are needed:

1. DB Secret: Adapt the Secret: tekton-results-postgres
2. Create the key/certificate and then the secret with the keys

## Setup Namespaced Tekton Results

To Setup the new User we created a esta\_tekton\_creator user with special privileges to create new users. This is a one time thing per database.   
We created this user already for the dev and the prod database.

CREATE ROLE esta\_tekton\_creator PASSWORD 'secret' NOSUPERUSER CREATEDB CREATEROLE INHERIT LOGIN;  
GRANT CONNECT ON DATABASE esta\_tekton TO esta\_tekton\_creator;  
GRANT ALL PRIVILEGES ON DATABASE esta\_tekton TO esta\_tekton\_creator WITH GRANT OPTION;

For the namespaced Tekton-Results Setup I created a repository to deploy tekton-results with the Helm Chart:

<https://code.sbb.ch/projects/KD_ESTA/repos/esta-tekton-results/browse>

Then I also created a new schema esta-tekton-predev and a user esta-tekton-predev which then needed to be granted with following SQL Statements:

DO  
$do$  
BEGIN  
 IF EXISTS (  
 SELECT FROM pg\_catalog.pg\_roles  
 WHERE rolname = 'test7\_tekton') THEN  
 RAISE NOTICE 'Role "test7\_tekton" already exists. Skipping.';  
 ELSE  
 CREATE ROLE test7\_tekton PASSWORD 'abcdefg' INHERIT LOGIN;  
 CREATE SCHEMA test7\_tekton;  
 GRANT CONNECT ON DATABASE esta\_tekton TO test7\_tekton;  
 ALTER ROLE test7\_tekton set SEARCH\_PATH = 'test7\_tekton';  
 PERFORM pg\_catalog.set\_config('search\_path', 'public', false);  
 grant usage on schema test7\_tekton to test7\_tekton;  
 GRANT ALL ON SCHEMA test7\_tekton TO test7\_tekton;  
 END IF;  
END  
$do$;

Ticket which describes the creation of user esta\_tekton\_creator: [DB-654](https://flow.sbb.ch/browse/DB-654) - Abrufen der Vorgangsdetails... STATUS

## Some usefull SQL commands

### Test with Index creations

I created several indexes:

CREATE INDEX idx\_results\_name ON esta\_tekton.results(name);  
CREATE INDEX idx\_records\_name ON esta\_tekton.records(name);  
CREATE INDEX idx\_records\_updated\_time ON esta\_tekton.records(updated\_time);  
  
But all queries used for fetching the PipelineRuns were not using the Indexes:

explain select rc.id, rc.name, rc.data, rc.type, rc.data from esta\_tekton.results as rs inner join esta\_tekton.records as rc on rs.name = rc.name order by rc.updated\_time desc;  
explain select rc.id, rc.name, rc.data, rc.type, rc.data from esta\_tekton.results as rs inner join esta\_tekton.records as rc on rs.name = rc.name order by rc.updated\_time desc;  
explain select rc.id, rc.name, rc.data, rc.type, rc.data from esta\_tekton.results as rs, esta\_tekton.records as rc where rs.name = rc.name order by rc.updated\_time desc limit 1000;  
  
So the indexes have no impact.

### Delete Schema and User

This SQL commands drop the schema and it's tables. And then also drop the user.

DROP SCHEMA <schema\_name> CASCADE;  
REVOKE CONNECT ON DATABASE esta\_tekton FROM <user\_name>;  
DROP ROLE <user\_name>;

### Reset Password of user

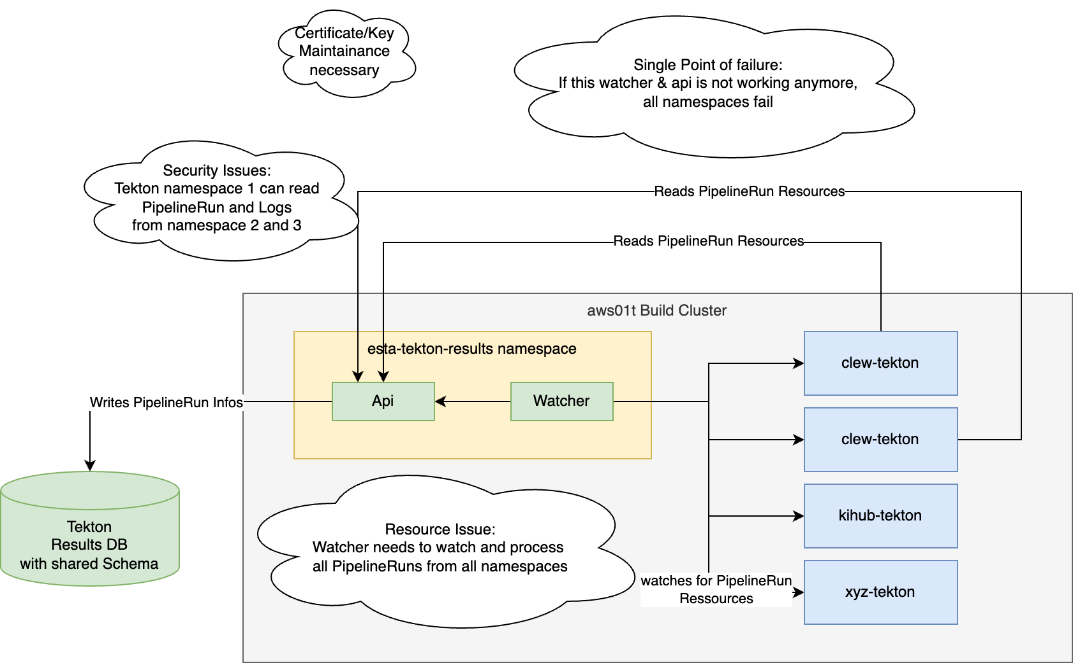
To reset the password of the Db User:

ALTER USER <user> WITH PASSWORD 'yourNewPassword';

## Tekton Result Setup Strategies

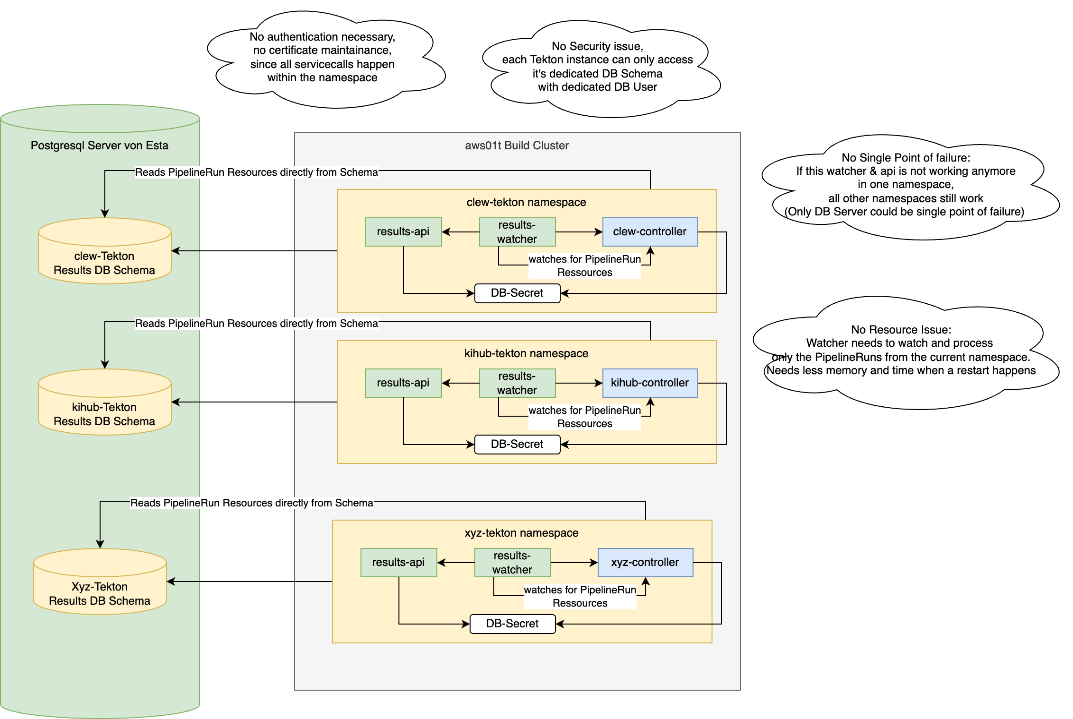
### Global Results

With the actual POC Setup we have a global Tekton Resource Watcher, which process all resources in all namespaces.



### Namespace Scoped Watcher

Another Setup would be, to install tekton results namespace scoped with its own schema per namespace. Also the watcher and api would run with namespace limited service account. This would be security wise a much better solution.   
Also stability wise we wouldn't have a single point of failure, except the database. The esta-tekton-pipeline-controller could actually access directly the database.

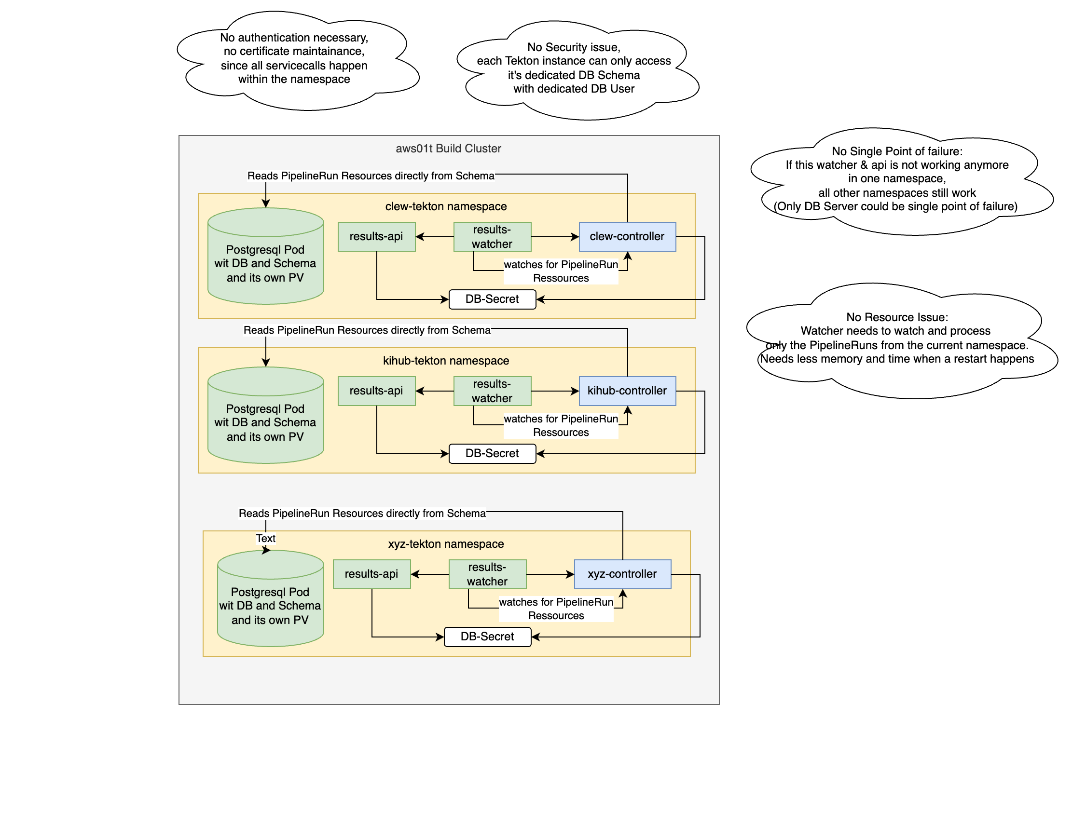


## DB Setup Options Tekton Result - Namespaced

### Option 1 Standard - Postgresql Setup per Namespace by Deployment

Here we would just setup a Postgresql Pod for each namespace with it's own PV which is then accessed by the Tekton-Results Api. Of course at first, this seems to be simple, but in case of a problem we need to manage 80-100 Postgresql instances.   
Also this solution will consume unnecessary lot's of hardware ressources and seems to me like unnecessary overhead.

|  |  |
| --- | --- |
| **Pro** | **Con** |
| Simple Setup by Deployment and ArgoCd | Managed by Esta |
| Ressource usage are billed directly to the project | Very high resource usage and overhead since 1 Postgresql instance per Tekton namespace |
| Reduced network traffic and latency for SQL Queries | Need to manage 80 Postgres Instances by ourselves |
|  | Postgres Knowhow needed |
|  | Updates need to be managed by Esta |



### Option 2 PGO Operator - Postgresql Setup per Namespace by Custom Resource (schematic picture same as above)

With this setup we would use the same operator as PGO uses to manage the Postgresql instances for all Tekton namespaces. But still we would have in each tekton-namespace it's own Postgresql instance.

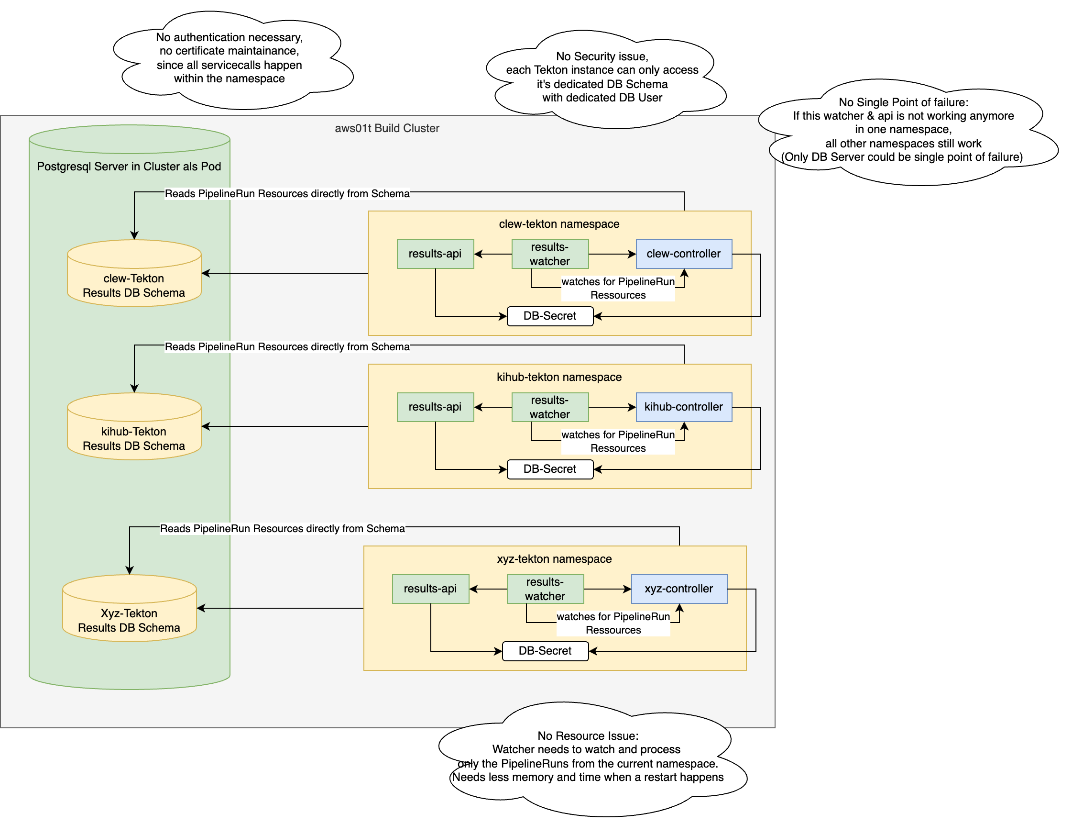
|  |  |
| --- | --- |
| **Pro** | **Con** |
| Simple Setup by Custom Resource and ArgoCD | Managed by Operator and Esta |
| Ressource usage are billed directly to the project | Very high resource usage and overhead since 1 Postgresql instance per Tekton namespace |
| Reduced network traffic and latency for SQL Queries | Need to manage 80 Postgres Instances by ourselves |
| Updates are handled by the Operator | Postgres Knowhow needed |
|  |  |

### Option 3 Single Cluster DB Instance - Self managed Postgresql Instance on Aws01t

With this setup we would have a self managed Postgresql DB instance on the aws01t Cluster. It can be selfmanaged or with the PGO Operator.   
But in the end, the setup happens by ourselves and is also managed by ESTA.   
The schematic setup is the same as with the Amazon RDS, just the instance is managed by ourselves. Also here we need to build up some DB Knowhow to manage, migrate, update the Postgresql DB.

So in the end we will have one Postgresql Server instance with one Database. The Esta-Tekton Setup process will create a new Scheme and DB User for each Tekton-Namespace.

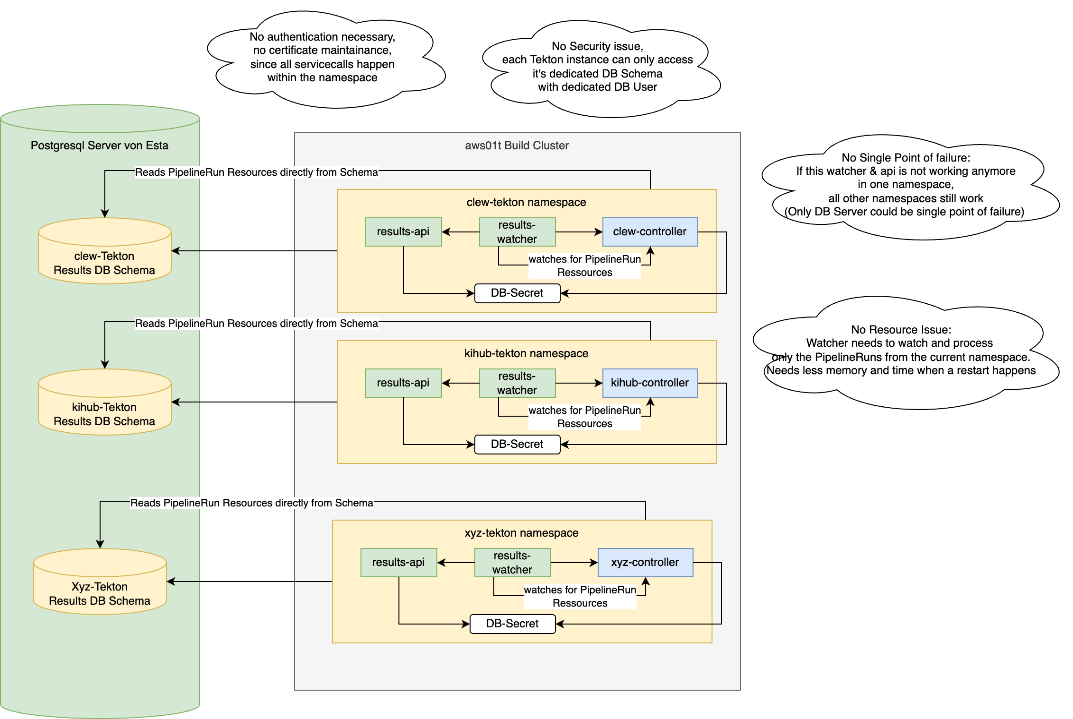
|  |  |
| --- | --- |
| **Pro** | **Con** |
| Simple Setup by DeploymentConfig or Operator | Managed by Esta |
| Reduced network traffic and latency for SQL Queries | Postgressql Knowhow needed |
| Low Resource usage with one PGSQL Instance and many schemas | Updates managed by Esta |
| Full controll over the PGSQL Instance | Shared costs |
|  | If this single instance is not working, access to historic Pipelineruns not possible |



### Option 4 Sigle RDS Instance - Multiple Schemas in one DB

The actual discussed solution and most probably most straight forward solution is to use one single RDS instance with many DB schemas and a DB User for each schema.

|  |  |
| --- | --- |
| **Pro** | **Cons** |
| Already setup | Network Latency since it runs outside of cluster |
| Separation of Data through Schemas and Users | Need to give rights to Admin user to create new users |
| PGSQL Instance managed by Amazon | PGSQL Instance managed by Amazon |
| Schema Setup through SQL already tested |  |
| Setup already tested |  |
| Relatively affordable costs |  |



## Setup Esta-Tekton RDS Schemas

At the moment my favorable solution is Option 4 with a single RDS DB instance and multiple Schemas. This solution makes it also easy to maintain this Database from outside, has low resource costs and also allows to run statistics over all Tekton-Namespaces without making cross Database Queries.

The schema setup seems also relatively easy to maintain and a possible solution I would like to show in following graphic. The idea behind is, that we have the full DB User which at the moment can create schemas and change users search path to the schema. At the moment the full DB User cannot create new users, but that can be solved. So to handle old and new tekton instances migrating to tekton-results, we would need to write an operator. This operator should do following things:

* Look for new Schema creation CR's on the tekton clusters
* Create a new DB User: xxx\_tekton
* Create a new Schmema: xxx\_tekton
* Grant created User to created schema
* Change searchpath of the default schema from the user to the newly created schema
* Create the DB secret in the xxx\_tekton namespace so tekton-results and the esta-tekton-pipeline-controller can access it

This Operator accesses the Database with a privileged user to create new users and schemas.

