Orbit

Onderzoek

Wat is de beste implementatiewijze voor de Orbit applicatie?

Naar	Luuk de Weijer
	Scrumble
Betreft	Orbit implementatie
Auteur(s)	Servi Huijbregts
Email	servi@scrumble.nl

Inleiding en doelstelling

Inleiding

Orbit is een applicatie die ervoor zorgt dat er meer inzicht komt in de verwerking van jobs en queues. Deze applicatie moet gemaakt worden omdat het huidige systeem dat dit faciliteert, Laravel Horizon, niet toereikend en gebruiksvriendelijk genoeg is.

Doelstelling

Het doel is om een eigen variant van Laravel Horizon te ontwikkelen die wel het inzicht en de gebruiksvriendelijkheid biedt wat de developers van Scrumble vereisen. Het is echter nog niet bekend wat de einddoelstellingen van het project nou precies inhouden, dit moet nog verder onderzocht/aangevuld worden als de mogelijkheden van het project verder onderzocht zijn.

Het doel van dit onderzoek staat echter wel vast. Dit doel is namelijk vaststellen welke implementatiemogelijkheden er zijn en op welke wijze Orbit geïmplementeerd kan worden. Moet dit door middel van een package implementatie of moet Orbit een op zichzelf draaiende applicatie worden waarin alle data van andere Scrumble applicaties verzameld wordt?

Onderzoeksvragen

Hoofdvraag	Wat is de beste implementatiewijze voor de Orbit applicatie?
Deelvragen	Welke implementatiemogelijkheden zijn er om het doel van Orbit te bereiken? Welke implementatie wijze zou het beste kunnen voldoen aan de niet-functionele requirements van het project?

Deelvraag 1

Vraag Welke implementatiemogelijkheden zijn er om het doel van Orbit te bereiken?

Resultaat Ik heb gesprekken gevoerd met 2 lead developers binnen Scrumble. Er zijn volgens deze lead developers twee mogelijke implementaties die interessant zijn. De ene optie is om Orbit te ontwikkelen als een op zichzelf staande applicatie waarin alle door Scrumble ontwikkelde applicaties toegevoegd kunnen worden. De andere optie is om Orbit te ontwikkelen als een PHP package, zodat deze geïmplementeerd kan worden in verschillende applicaties van Scrumble en dus per applicatie kan draaien.

De gehele applicatie zou een applicatie zijn die is gebouwd vanuit het baseproject van Scrumble. Dit baseproject bestaat uit een Laravel(PHP) backend en een React(Javascript/typescript) front end. Hierin zouden dan functionele requirements zoals een algeheel dashboard en het toevoegen en beheren van applicaties mogelijk zijn.

De package implementatie zou per project geïmplementeerd moeten worden en per project een dashboard geven. Het aanmaken en beheren van applicaties is hierbij dan niet nodig en mogelijk, dit zou eventueel vervangen kunnen worden door het aanmaken van eigen filter functionaliteit waarbij je kunt aangeven dat je bijvoorbeeld alle pending jobs van een bepaalde tag wilt zien. Zoals ik al aangeef is dit een hypothetische situatie.

Conclusie

Uit het resultaat van de vraag blijkt dat er twee implementatiemogelijkheden zijn om het doel van Orbit te bereiken: als een op zichzelf staande applicatie of als een package die geïmplementeerd kan worden in verschillende applicaties.

De eerste optie zou alle applicaties bevatten en vanuit het baseproject van Scrumble ontwikkeld worden, met een Laravel(PHP) backend en een React(Javascript/typescript) frontend. Het biedt de mogelijkheid om functionele requirements zoals een dashboard en applicatiebeheer te integreren.

De tweede optie zou per project geïmplementeerd worden en een dashboard per project bieden. Applicatiebeheer is hierbij niet nodig en kan vervangen worden door het aanmaken van eigen filter functionaliteit.

Het lijkt erop dat beide implementatiemogelijkheden haalbaar zijn, maar de keuze hangt af van de specifieke vereisten en doelstellingen van het project.

Deelvraag 2

Vraag

Welke implementatie wijze zou het beste kunnen voldoen aan de niet-functionele requirements van het project?

Resultaat

De niet functionele requirements van het project zijn performance en gebruiksvriendelijkheid, zoals in het projectplan te lezen is.

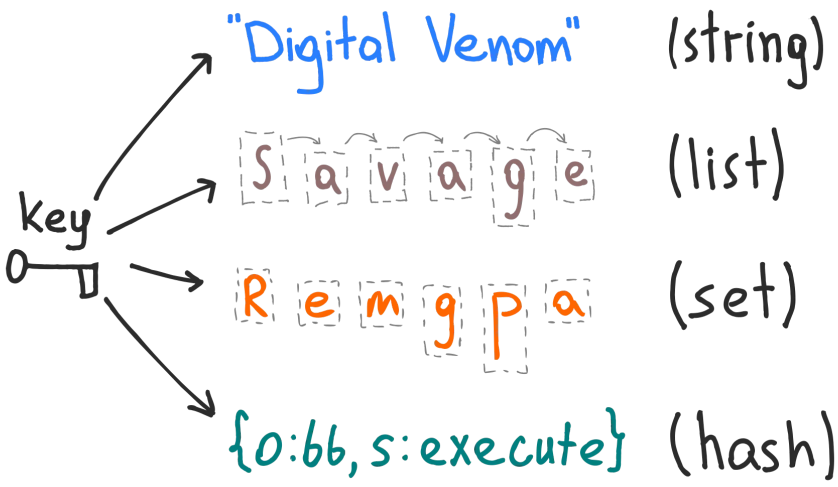
De grootste verschillen tussen de twee implementatiemogelijkheden (Package of applicatie) zijn usability, security en performance.

Het verschil in usability komt door de mogelijkheid om bij een volledige applicatie redis keys toe te voegen om te monitoren, als je het systeem als package released is het gemaakt voor één specifieke applicatie en zul je dus per applicatie moeten navigeren naar de pagina waarop het dashboard te zien krijgt. Deze kennis heb ik opgedaan door het maken van een proof of concept waarbij het uitgangspunt was om een gehele applicatie te maken waarin alle Scrumble applicaties verwerkt zaten.

Het verschil in security zit hem voornamelijk, dat in het geval van een datalek, alle redis database gegevens op straat zouden kunnen komen te liggen. Dit is natuurlijk af te vangen met goede authenticatie middleware. Uiteindelijk is het risico er altijd dat iemand de authenticatie omzeilt, maar in het geval van een geïmplementeerde package moet je bij iedere applicatie de authenticatie omzeilen om iedere aparte redisconnectie te ontdekken. Als alle applicaties binnen één applicatie verzameld zouden staan is er dus in het geval van een datalek het gevaar dat alle Redis gegevens van al deze systemen samen op straat komen te liggen.

Het verschil in performance is gebaseerd op het aantal Redis records die opgehaald moeten worden door de applicatie. Als je een dashboard wilt creëren waarop data van alle applicaties die door Scrumble gereleased zijn wilt monitoren, dan kun je je inbeelden dat de call hiervoor een lange tijd zal innemen. Een dashboard per applicatie lijkt daarin sneller te werken.

Als we verder in de documentatie van Redis(deze is te vinden op https://redis.io/docs/) duiken, komen we de “Time complexity” noemer tegen voor iedere command die uitgevoerd kan worden. Er zijn verschillende datatypen in Redis. Een overzicht van de mogelijkheden zie je in onderstaande afbeelding:



Een key kan een string, een list, een set of een hash bevatten. Zoals de afbeelding ook illustreert zien deze data types er allemaal anders uit. Een string is een standaard stuk tekst. Een list is een opsomming van geordende elementen. Een set is een collectie van unieke, niet gesorteerde string elementen. En ten slotte is een hash te vergelijken met een JSON, een samenstelling van velden en waarden.

Voor al deze data types zijn er commando’s en degene die wij moeten gebruiken zijn voornamelijk de HGET, LRANGE, KEYS commando’s.

Commando	Wat doet het?
HGET	Haalt een specifieke hashmap key op.
LRANGE	Haalt een bepaalde range binnen

	een lijst op. (Bijvoorbeeld index 100 tot 150)
KEYS	Haalt alle keys uit de redis database als strings op zonder onderliggende waardes.

De "time complexity" wordt aangeduid door middel van een grote-O-notatie. Zo heeft HGET een "time complexity" waarde van O(1), LRANGE O(S+N), KEYS O(N).

"Time complexity"-waarde	Uitleg
O(1)	Hierbij is de tijd onafhankelijk van de grootte van de dataset.
O(S+N)	Lineaire groei in tijd op basis van de afstand van het begin tot de startpositie in de lijst en de N het aantal elementen die binnen deze radius zitten.
O(N)	Lineaire groei in tijd op basis van het aantal (in dit geval keys) in de database.

Als je deze 3 waardes in een tabel zou vergelijken met elkaar dan zie je dat de 0(1) waarde het snelste is omdat deze onafhankelijk van de grootte van het aantal elementen in een dataset dezelfde tijd gebruikt. De O(N) waarde is daarna de snelste omdat deze een lineaire groei in tijdsbesteding heeft op basis van een aantal elementen in de dataset. De O(S+N) waarde is de minst snelle omdat, ondanks dat deze tevens een lineaire groei in tijdsbesteding heeft, deze 2 factoren heeft die bij elkaar opgeteld worden in plaats van 1 waarvan de N gelijk zal zijn aan de N in de O(N) waarde..

Om de snelste performance te behalen wil je dus gebruikmaken van de HGET functionaliteit. In het geval van de Scrumble applicaties zien de jobs die in de Redis database staan zijn van het datatype “hash” en zien er als volgt uit:

Field	Value

payload	{
	"uuid": "f7c4285f-84fb-4bc2-ad32-50325ec529dd",
	"displayName": "App\\Jobs\\SomeJob",
	"job": "Illuminate\\Queue\\CallQueuedHandler@call",
	"maxTries": null,
	"maxExceptions": null,
	"failOnTimeout": false,
	"backoff": null,
	"timeout": null,
	"retryUntil": null,
	"data": {
	"commandName": "App\\Jobs\\SomeJob",
	"command": "O:16:\"App\\Jobs\\SomeJob\":1:
	{s:25:\"\u0000App\\Jobs\\SomeJob\u0000theData\";a:7:
	{s:5:\"array\";a:3:
	{i:0;i:1;i:1;i:2;i:2;i:3;}s:7:\"boolean\";b:1;s:5:\"color\";s:4:\"gold\";s:4:
	\"null\";N;s:6:\"number\";i:123;s:6:\"object\";O:8:\"stdClass\":2:
	{s:1:\"a\";s:1:\"b\";s:1:\"c\";s:1:\"d\";}s:6:\"string\";s:11:\"Hello
	World\";}}"
	},
	"id": "f7c4285f-84fb-4bc2-ad32-50325ec529dd",
	"attempts": 0,
	"type": "job",
	"tags": [],
	"silenced": false,
	"pushedAt": "1678109835.9234"
	}

id	f7c4285f-84fb-4bc2-ad32-50325ec529dd
queue	default
connection	redis
status	pending
updated_at	1678109835.9236
name	App\Jobs\SomeJob
created_at	1678109835.9236

Omdat de elementen van het datatype “hash” zijn kun je de HGET functionaliteit uitvoeren wat dus qua performance de snelste resultaten oplevert.

Dan kom je bij de volgende zaak die invloed zou kunnen hebben op de performance van de applicatie en dat is het aantal records waarover de applicatie heen moet lezen om een bepaalde dataset terug te krijgen. Laravel horizon lost dit probleem op door de paginaten, hierbij voer je de call uit per x aantal elementen.

```php
    Get all of the failed jobs.

    Parameters: Request $request
    Returns:    array
public function index(Request $request)
{
    $jobs = ! $request->query( key: 'tag')
            ? $this->paginate($request)
            : $this->paginateByTag($request, $request->query( key: 'tag'));

    $total = $request->query( key: 'tag')
            ? $this->tags->count( tag: 'failed:'.$request->query( key: 'tag'))
            : $this->jobs->countFailed();

    return [
        'jobs' => $jobs,
        'total' => $total,
    ];
}
```

Hierbij zie je dat de jobs worden gepaginate, dit wordt uitgevoerd door een van de twee volgende functies, als er geen tag meegegeven wordt:

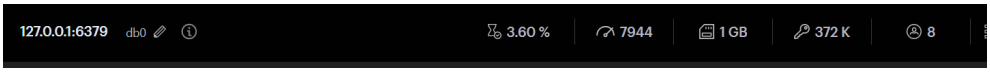
```php
    Paginate the failed jobs for the request.

    Parameters: Request $request
    Returns:    Collection
protected function paginate(Request $request)
{
    return $this->jobs->getFailed( afterIndex: $request->query( key: 'starting_at') ?: -1)->map(function ($job) {
        return $this->decode($job);
    });
}
```


En als er wel een tag meegegeven wordt:

```
Paginate the failed jobs for the request and tag.

Parameters: Request $request
            string $tag
Returns:    Collection

protected function paginateByTag(Request $request, $tag)
{
    $jobIds = $this->tags->paginate(
        tag: 'failed:'.$tag,  startingAt: ($request->query( key: 'starting_at') ?: -1) + 1,  limit: 50
    );

    $startingAt = $request->query( key: 'starting_at',  default: 0);

    return $this->jobs->getJobs($jobIds, $startingAt)->map(function ($job) {
        return $this->decode($job);
    });
}
```

Hiermee zorgt Laravel Horizon ervoor dat de calls niet te groot worden en weinig tijd in beslag nemen. Een van de functional requirements van dit project is echter om door alle jobs heen te kunnen filteren met specifieke parameters. Dit is dus niet mogelijk als je maar 50 elementen inlaadt. Je moet er dus voor zorgen dat ofwel je dataset klein genoeg is om in 1x in te laden, of je moet een functie schrijven die zeer snel door alle redis database entiteiten heen kan gaan.

De grootte van de dataset is afhankelijk van de hoeveelheid jobs die er in de applicatie per tijdsperiode uitgevoerd worden. De grootte van de dataset is afhankelijk van het aantal gebruikers en het aantal instanties van asynchrone logica van applicaties. Een doorsnee applicatie waarin veel asynchrone logica zit van Scrumble verstuurd ruim 400000 jobs per dag. Om te kijken hoe groot een dataset zou kunnen zijn om nog steeds juist te functioneren heb ik 400000 jobs gedispatched naar redis. Dit zorgt voor een database grootte van 1 gigabyte:



Als ik hier een simpele functie op uitvoer zoals bijvoorbeeld het ophalen van alle keys en een hgetall op deze keys uit te voeren duurt dit voor

100 Keys	0,494 seconden
1000 Keys	0,829 seconden
10000 Keys	3,271 seconden
100000 Keys	39,310 seconden
420000 Keys	Kan niet uitgevoerd worden omdat de memory size exhausted is.

Ik kan dus geen vergelijking maken tussen een dataset van 1 applicatie of een dataset van 30+ verschillende applicaties aangezien de tijd voor 100000 keys al te lang is om te gebruiken. Ik zal dus een manier moeten vinden om de dataset op een manier te behandelen waarbij zowel de functionele als niet functionele requirements behaald worden.

Conclusie

De implementatiewijze die het beste zou kunnen voldoen aan de niet-functionele requirements van het project, hangt af van de specifieke eisen en omstandigheden. De implementatiewijze waarbij Orbit als package wordt uitgebracht, kan minder gebruiksvriendelijk zijn omdat gebruikers per applicatie moeten navigeren om het dashboard te bekijken, terwijl bij een volledige applicatie redis keys kunnen worden toegevoegd om te monitoren waardoor alles op 1 dashboard staat, dit zal echter ten koste gaan van de performance. Wat betreft security kan er een risico zijn van gegevens lekken bij zowel een geïmplementeerde package als bij een gehele applicatie, maar dit kan worden afgevangen met goede authenticatie middleware. In het geval dat er toch een datalek is, is een package implementatie veiliger omdat er dan maar van 1

database gegevens lekken. Wat betreft performance kan HGET de snelste resultaten opleveren voor het opvragen van datasets uit de Redis database. Laravel Horizon lost het probleem van het aantal records op door paginering toe te passen, maar dit kan beperkingen opleggen aan het filteren van de gegevens en de gebruikerservaring door lange wachttijden met betrekking tot laden van gegevens. Het is dus zaak om een andere methodiek te gaan onderzoeken waarop grote datasets wel opgehaald kunnen worden, dit zal resulteren in een nieuw proof of concept.

APA Bronvermelding

HGET. (n.d.). Redis. https://redis.io/commands/hget/

HGETALL. (n.d.). Redis. https://redis.io/commands/hgetall/

KEYS. (n.d.). Redis. https://redis.io/commands/keys/

Laravel - The PHP Framework For Web Artisans. (n.d.). https://laravel.com/docs/10.x/horizon

LRANGE. (n.d.). Redis. https://redis.io/commands/lrange/