Magento 2 Enterprise Edition

Important developer concepts

DevelopersParadise

2016 / Opatija / Croatia

Anton Kril

Lead Architect - Magento

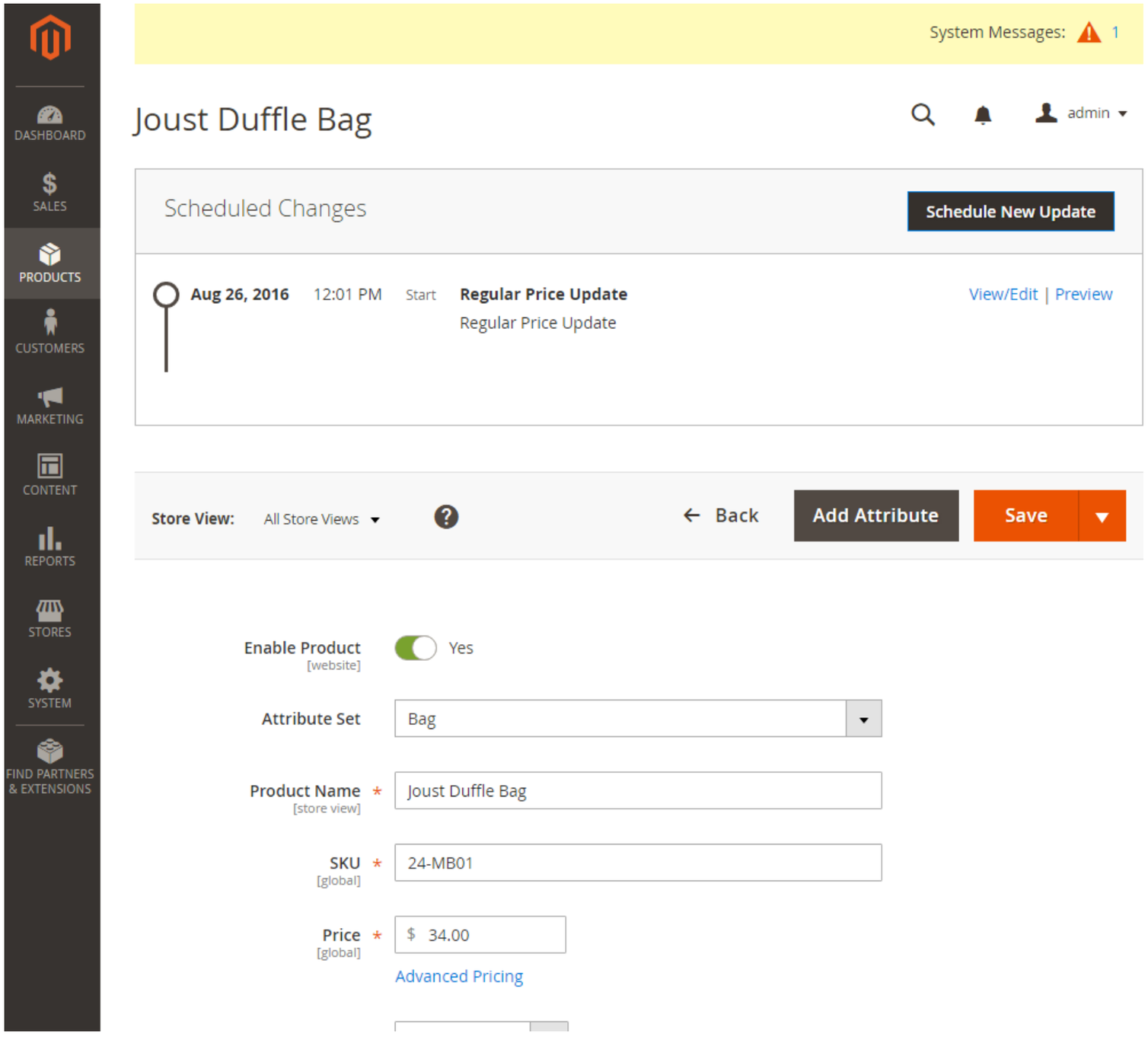
@AntonKril

http://antonkril.github.io

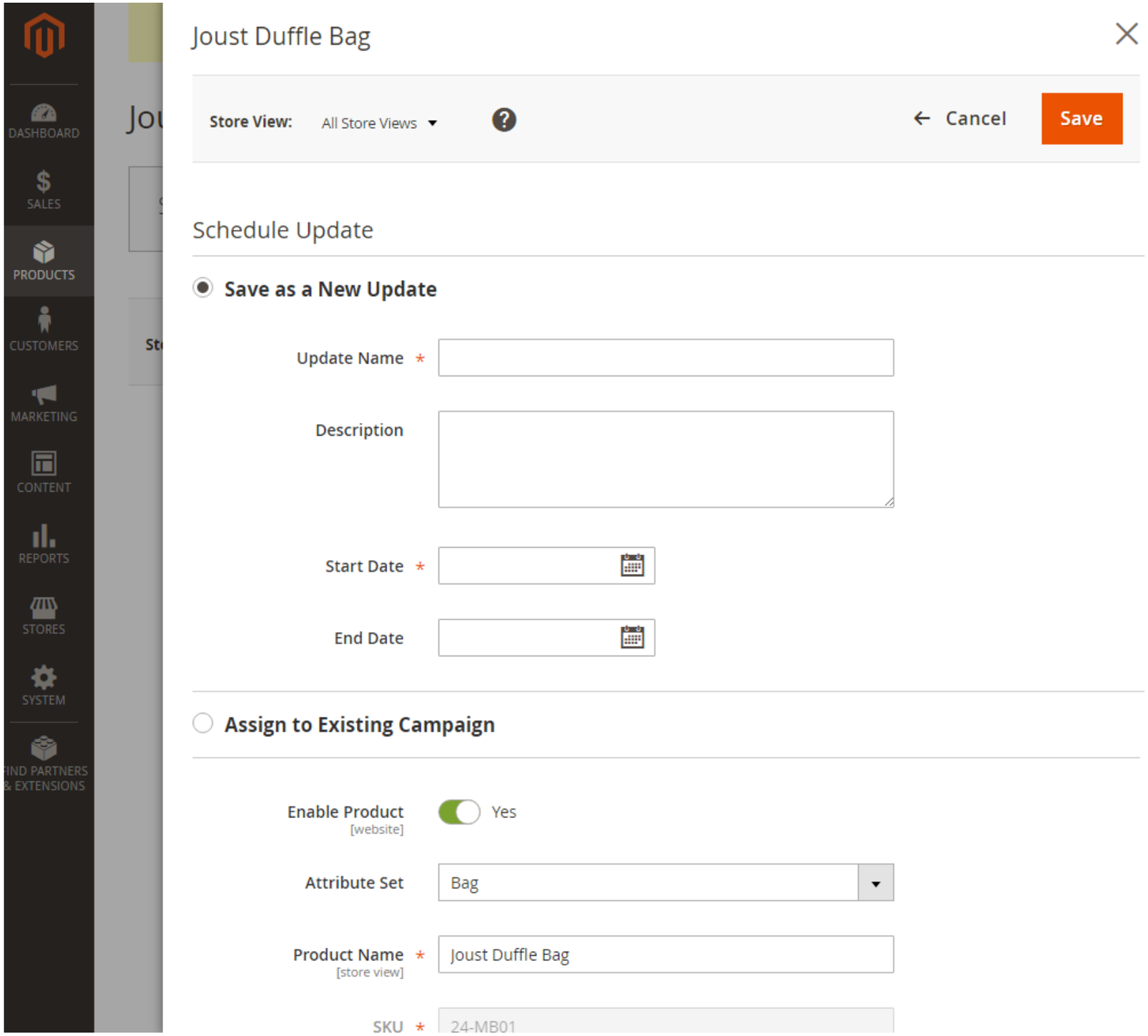
Staging

1. Staging
2. BundleStaging
3. CatalogImportExportStaging
4. CatalogInventoryStaging
5. CatalogRuleStaging
6. CatalogStaging
7. CatalogUrlRewriteStaging
8. CheckoutStaging
9. CmsStaging
10. ConfigurableProductStaging
11. DownloadableStaging
12. GiftCardStaging
13. GiftMessageStaging
14. GiftWrappingStaging
15. GoogleOptimizerStaging
16. GroupedProductStaging
17. LayeredNavigationStaging
18. MsrpStaging
19. PaymentStaging
20. ProductVideoStaging
21. ReviewStaging
22. RmaStaging
23. SalesRuleStaging
24. SearchStaging
25. WeeStaging

Staging. Scheduled changes



Staging. Change editing



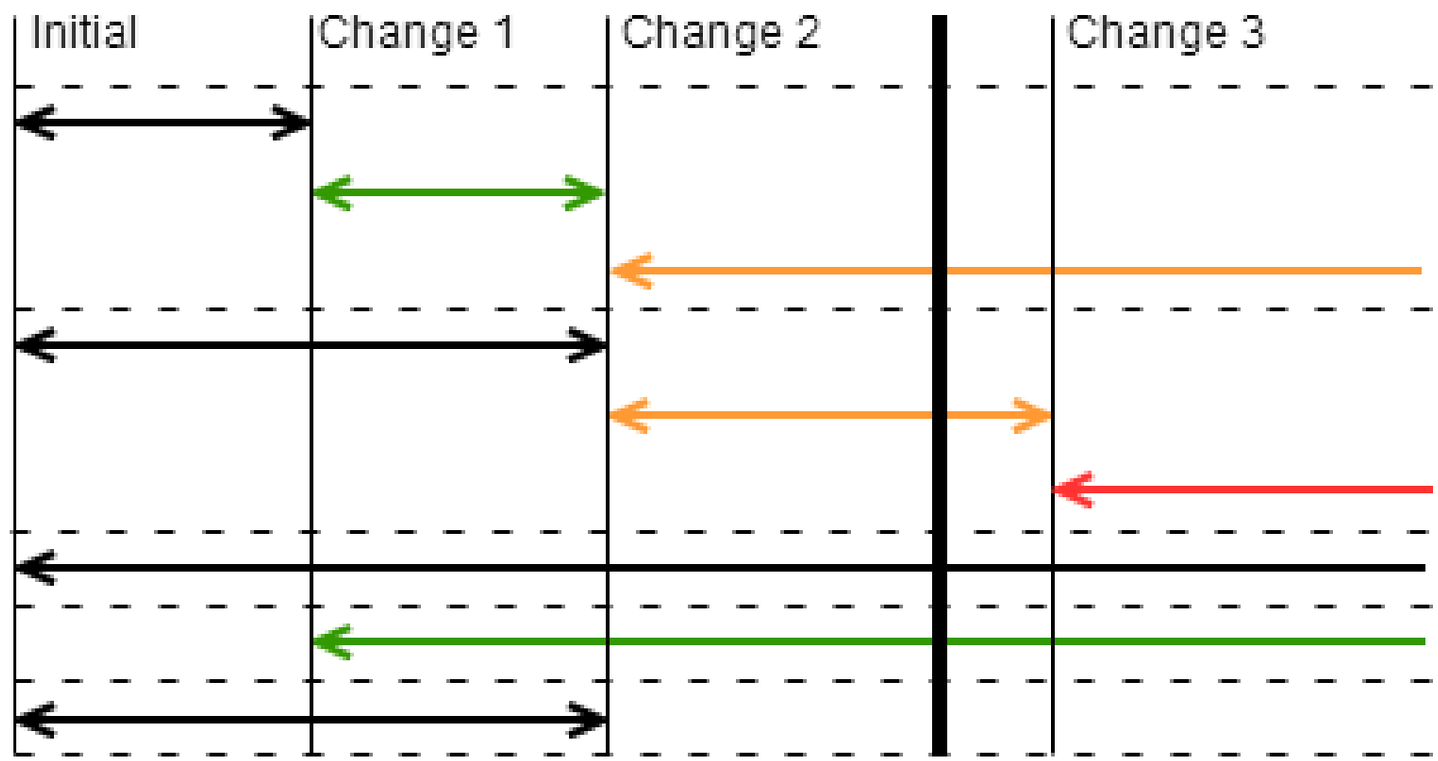
Staging. Requirements

	Campaign A	Campaign B
Entity 1 - Property 1 - Property 2	Entity 1 - Property 1A - Property 2	Entity 1 - Property 1 - Property 2B
Entity 2 - Property 1 - Property 2	Entity 2 - Property 1 - Property 2	Entity 2 - Property 1 - Property 2
	Entity 3 - Property 1 - Property 2	Entity 3 - Property 1 - Property 2

Staging. Database structure

```
SELECT * FROM entity_table WHERE created_in <= $staging_version AND updated_in >
$staging_version
```

	C	U
Entity A	0	1
Entity A`	1	2
Entity A``	2	
Entity B	0	2
Entity B`	2	3
Entity B``	3	
Entity C	0	
Entity D	1	
Entity E	0	2



Preview Snapshot
(Created <= Current AND Updated > Current)

Update Repository

```
namespace Magento\Staging\Api;
use \Magento\Framework\Api\SearchCriteriaInterface;

interface UpdateRepositoryInterface
{
    public function getList(SearchCriteriaInterface $criteria);

    public function get($id);

    public function delete(\Magento\Staging\Api\Data\UpdateInterface $entity);

    public function save(\Magento\Staging\Api\Data\UpdateInterface $entity);

    public function getVersionMaxIdByTime($timestamp);
}
```


Staging

```
namespace Magento\CatalogStaging\Model;
class ProductStaging implements ProductStagingInterface
{
    public function schedule(\Magento\Catalog\Api\Data\ProductInterface $product, $version)
    {
        return $this->entityManager->save(
            $product,
            \Magento\Catalog\Api\Data\ProductInterface::class,
            [
                'store_id' => $this->storeManager->getStore()->getId(),
                'created_in' => $version
            ]
        );
    }

    public function unschedule(\Magento\Catalog\Api\Data\ProductInterface $product, $version)
    {
        return $this->entityManager->delete(
            $product,
            \Magento\Catalog\Api\Data\ProductInterface::class,
            [
                'store_id' => $this->storeManager->getStore()->getId(),
                'created_in' => $version
            ]
        );
    }
}
```

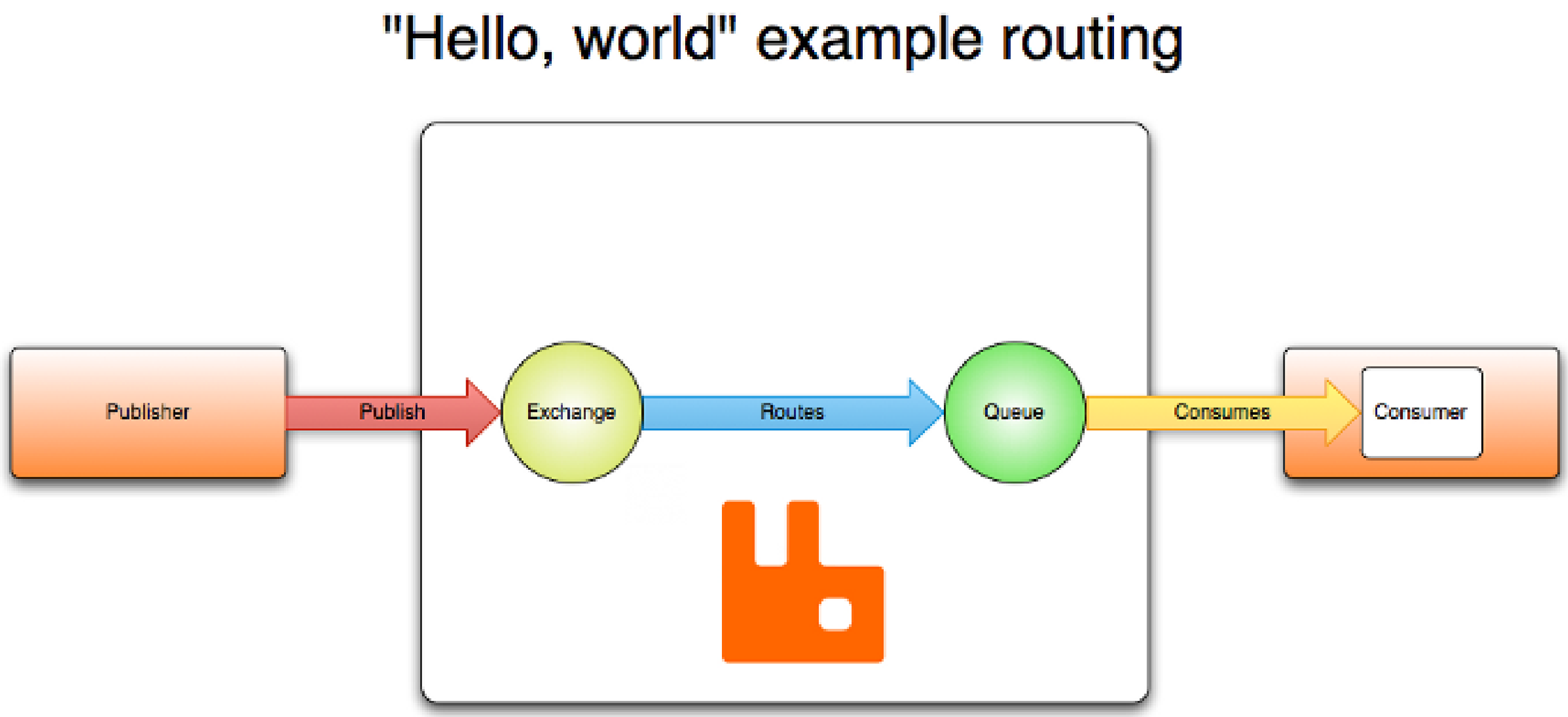
Queue

1. Magento\Framework\MessageQueue
2. Amqp
3. MessageQueue
4. ScalableInventory

Job Queue Framework

- An AMQP protocol implementation for Rabbit MQ & MySQL
- Configuration defines queues, exchanges; topic-based routing for messages
- Interfaces are available to send messages and to attach message handlers
- CLI command starts consumers
- Used for asynchronous stock updates
- Easy scalability for system tasks

Queue


"Hello, world" example routing
Publisher
Publish
Exchange
Routes
Queue
Consumes
Consumer

Queue

```
<topic publisher="default" name="inventory.counter.updated”
    schema="Magento\ScalableInventory\Api\Counter\ItemsInterface" />

<consumer name="inventoryQtyCounter" queue="inventory_qty_counter_queue”
    connection="amqp” executor="Magento\Framework\MessageQueue\BatchConsumer"
    class="Magento\ScalableInventory\Model\ResourceModel\QtyCounterConsumer"
    method="processMessage"/>

<bind queue="inventory_qty_counter_queue”
    exchange="magento" topic="inventory.counter.updated" />
```

Search

1. AdvancedSearch
2. Elasticsearch
3. Solr

Search

Filters

1. Exact matching
2. Binary yes/no searches

Queries

1. Full text searches
2. Relevance sorting

Aggregation

1. Faceted Search

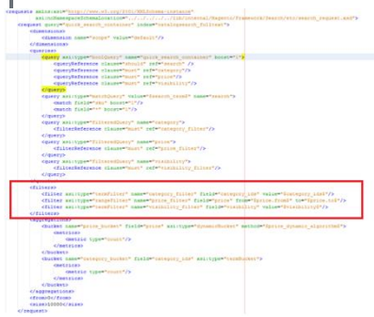
Declarative Search. Queries

app\code\Magento\CatalogSearch\etc\search_request.xm
l

```
<dimensions>
    <dimension name="scope" value="default"/>
</dimensions>
<queries>
    <query xsi:type="boolQuery" name="quick_search_container" boost="1">
        <queryReference clause="should" ref="search" />
        <queryReference clause="must" ref="category"/>
        <queryReference clause="must" ref="price"/>
        <queryReference clause="must" ref="visibility"/>
    </query>
    <query xsi:type="matchQuery" value="$search_term$" name="search">
        <match field="sku"/>
        <match field="*"/>
    </query>
    <query xsi:type="filteredQuery" name="category">
        <filterReference clause="must" ref="category_filter"/>
    </query>
    <query xsi:type="filteredQuery" name="price">
        <filterReference clause="must" ref="price_filter"/>
    </query>
    <query xsi:type="filteredQuery" name="visibility">
        <filterReference clause="must" ref="visibility_filter"/>
    </query>
</queries>
```

Declarative Search. Filters

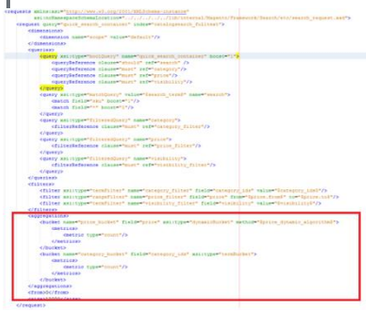
app\code\Magento\CatalogSearch\etc\search_request.xm
l



```
<filters>
    <filter xsi:type="termFilter" name="category_filter" field="category_ids"
            value="$category_ids$"/>
    <filter xsi:type="rangeFilter" name="price_filter" field="price"
            from="$price.from$" to="$price.to$"/>
    <filter xsi:type="termFilter" name="visibility_filter" field="visibility"
            value="$visibility$"/>
</filters>
```

Declarative Search. Aggregations

app\code\Magento\CatalogSearch\etc\search_request.xm
l

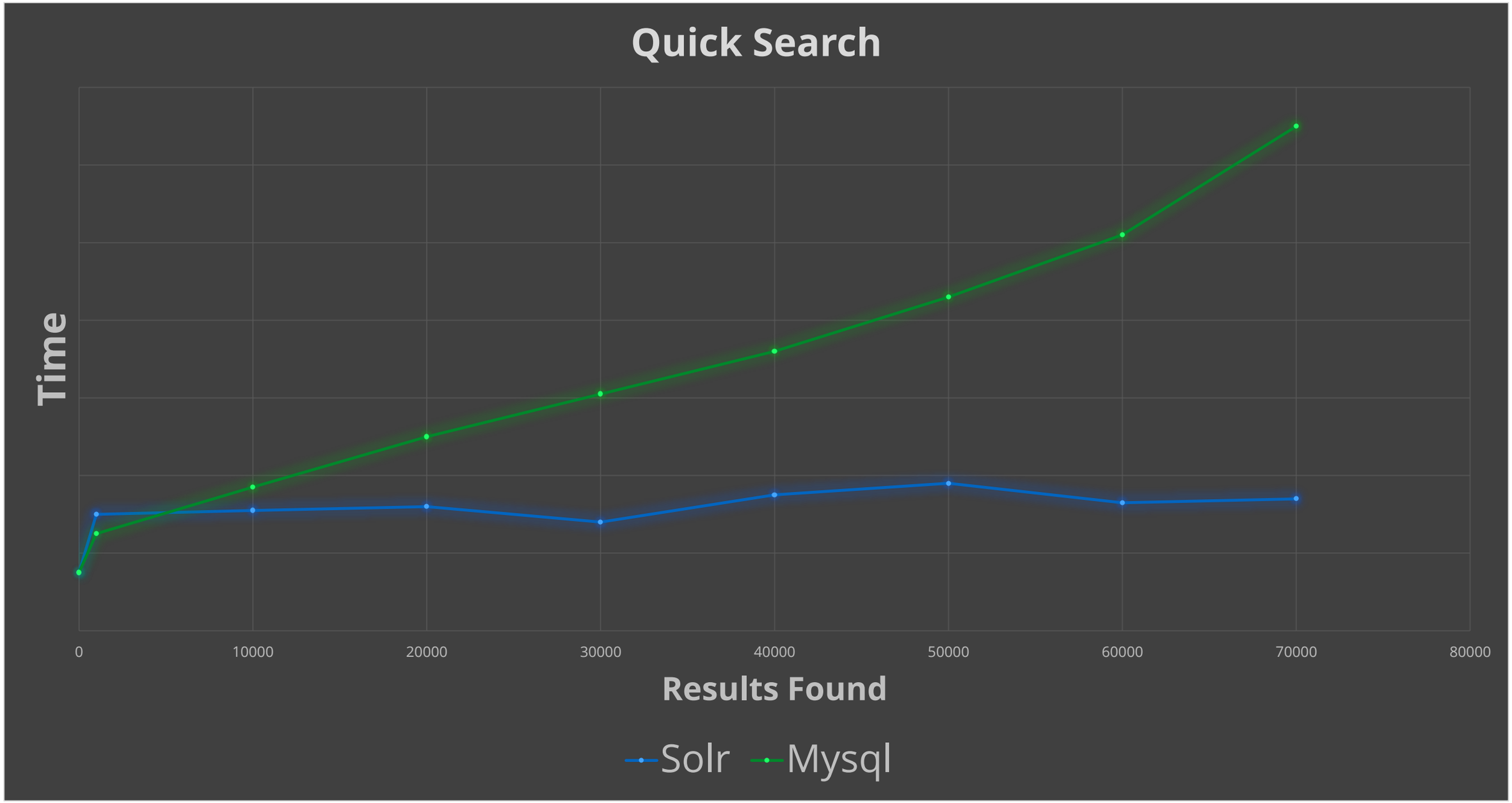


```
<aggregations>
    <bucket name="price_bucket" field="price" xsi:type="dynamicBucket"
        method="$price_dynamic_algorithm$">
        <metrics>
            <metric type="count"/>
        </metrics>
    </bucket>
    <bucket name="category_bucket" field="category_ids" xsi:type="termBucket">
        <metrics>
            <metric type="count"/>
        </metrics>
    </bucket>
</aggregations>
```

Search Service Contract

```php
namespace Magento\Framework\Api\Search;
/**
* Search API for all requests
*
* @api
*/
interface SearchInterface
{
    public function search(
        \Magento\Framework\Api\Search\SearchCriteriaInterface $searchCriteria
    );
}
```

Performance



** Don't consider values. Only trend is important*

Magento meets Elasticsearch

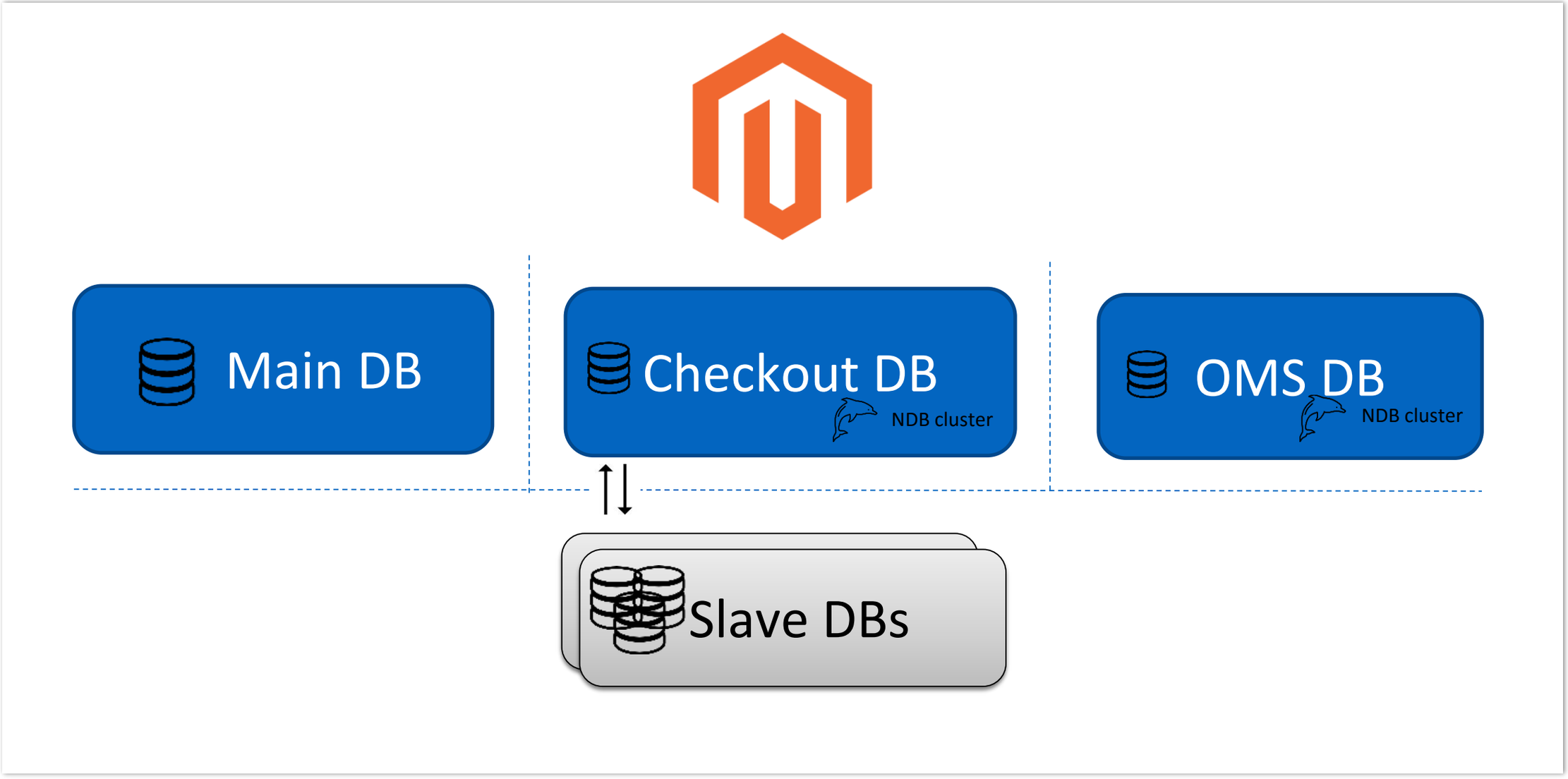
elastic

1.7. and 2.* versions supported*

Database Split

1. ScalableCheckout
2. ScalableOMS

Database Split



Database Split

Split Checkout:

```
magento setup:db-schema:split-quote --host="checkout-db-
host" --dbname="checkout" --username="checkout" --
password="123123q"
```

Split OMS:

```
magento setup:db-schema:split-sales --host="sales-db-host" --
dbname="sales" --username="sales" --password="123123q"
```

Database Connections

ResourceConnections

- Write Connection on GET requests rarely makes sense
- Write connection is mostly needed on POST requests

Database Connections

- Framework decides which connection to use
- Allow to explicitly ask for connection type
- Enterprise Allows to configure Slave read-only connections

```
'db' => [
    'connection' => [
        'default' => [
            'host' => 'default-master-host',
            // ...
        ],
    ],
    'slave_connection' => [
        'default' => [
            'host' => 'default-slave-host',
            'dbname' => 'magento',
            // ...
        ],
    ],
    'table_prefix' => '',
],
```


Thank You!

DevelopersParadise

2016 / Opatija / Croatia