MAKAIRA.io

Extend shopware flowbuilder

Add own triggers and actions

Ein Unternehmen der ECOMMERCE ONE

Welcome

- **Miriam Müller**

- Makaira GmbH
- Senior Fullstack Developer
- 15 years experience with ecommerce projects
- Shopware since version 4.6
- ******

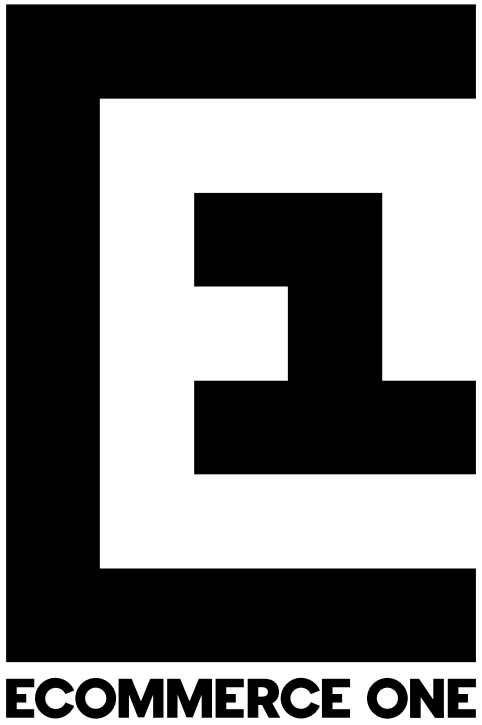
MAKAIRA.io

Welcome

- **Miriam Müller**

- Makaira GmbH
- Senior Fullstack Developer
- 15 years experience with ecommerce projects
- Shopware since version 4.6
- Winner community challenge cup 2024

MAKAIRA.io



Afterbuy™

DreamRobot

baygraph

gambio

MAKAIRA.io

marmalade

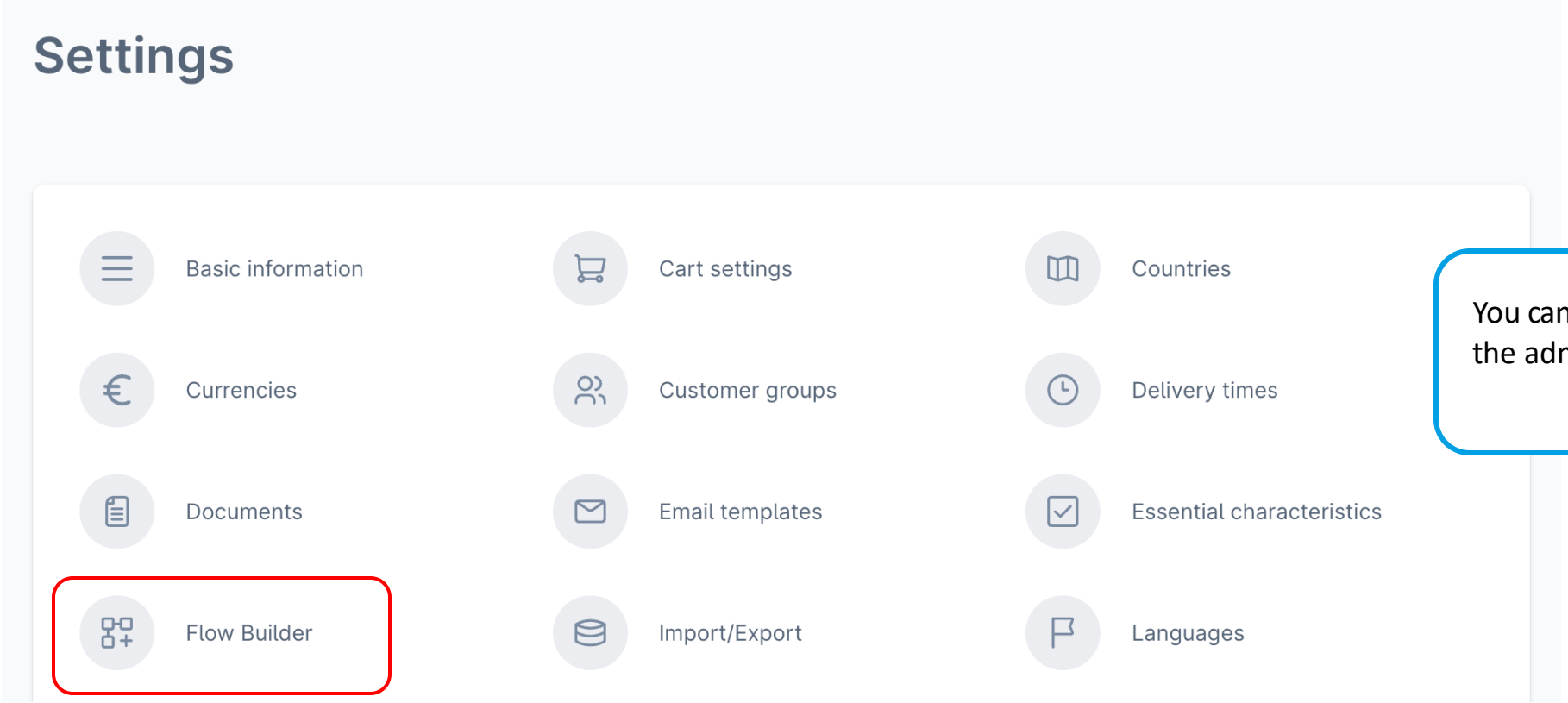
Agenda

- **Flowbuilder introduction**
- **Example description**
- **Example part 1: add custom trigger for stock changes**
- **Example part 2: add custom field action for product entity**
- **Example part 3: add custom rule for product custom fields**
- **Example part 4: add custom subscriber list for mail action**
- **Example part 5: add custom trigger for custom entity with store**

Flowbuilder introduction

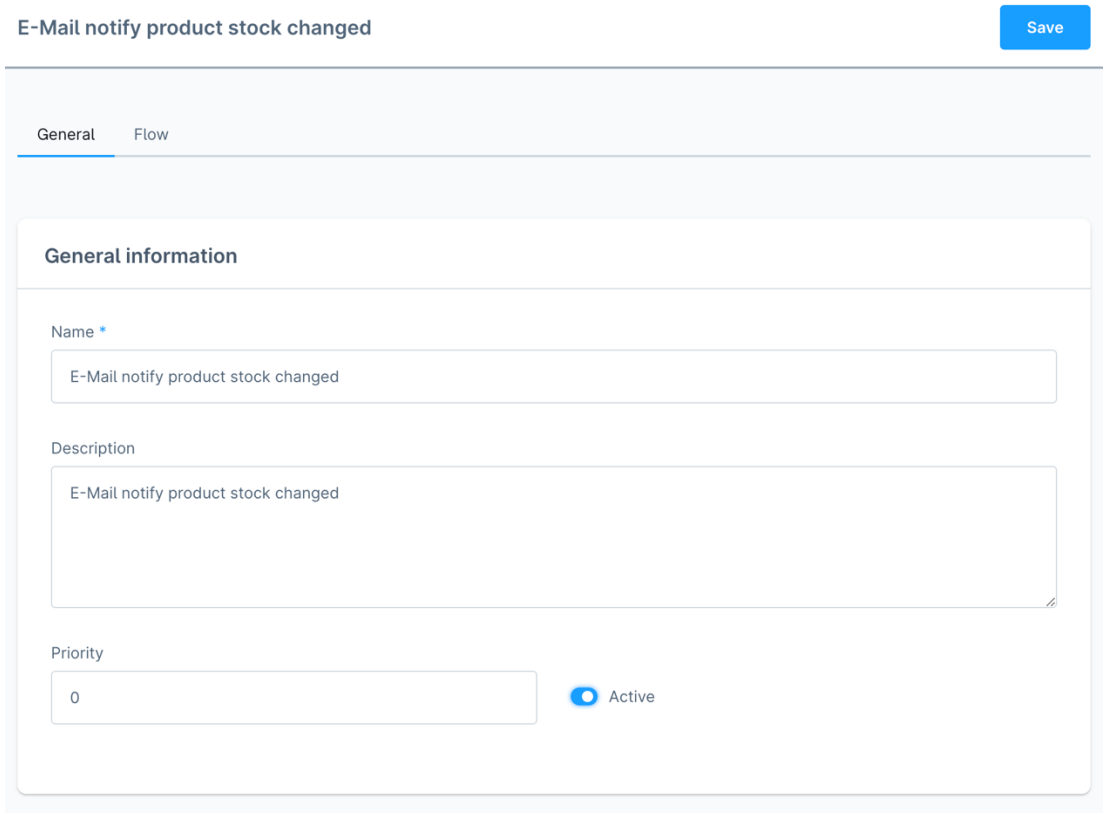
- **No-Code Approach / Visual Workflow Design:**
  Drag-and-drop interface for creating workflows without programming skills.
- **Real-Time Execution:**
  Automations run instantly when triggers (e.g., order placement, payment status) occur.
- **Conditional Logic:**
  Customize workflows with rule builder conditions, triggering actions only when specific criteria are met.
- **Extensive list of existing triggers and actions:**
  An extensive list of triggers and actions is already included. E.g. order placement, payment status, customer registration and many more.
- **Extendable:**
  Easy to extend with your own triggers and actions.

Flowbuilder introduction



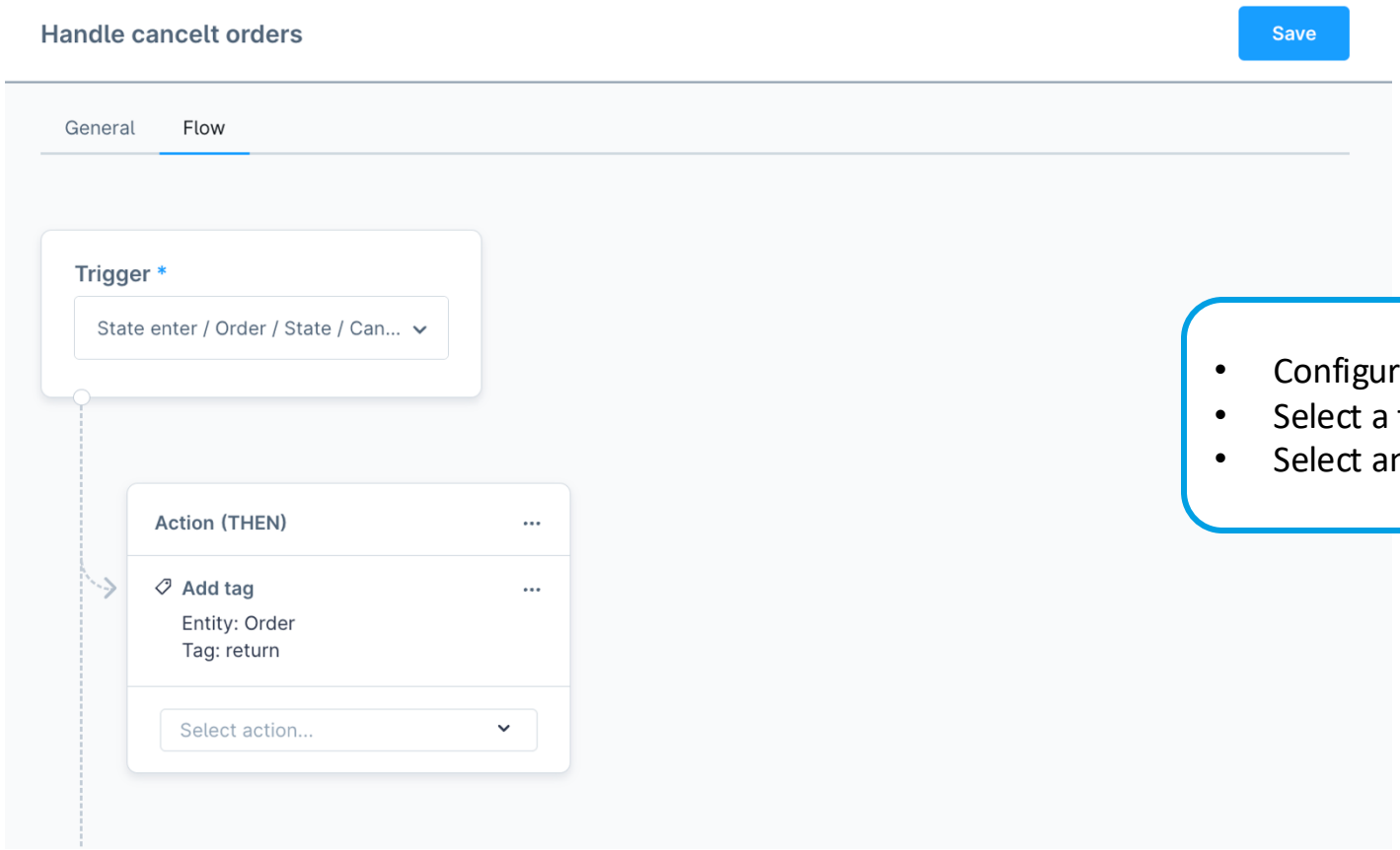
You can find the flowbuilder in the administration in the store settings.

Flowbuilder introduction



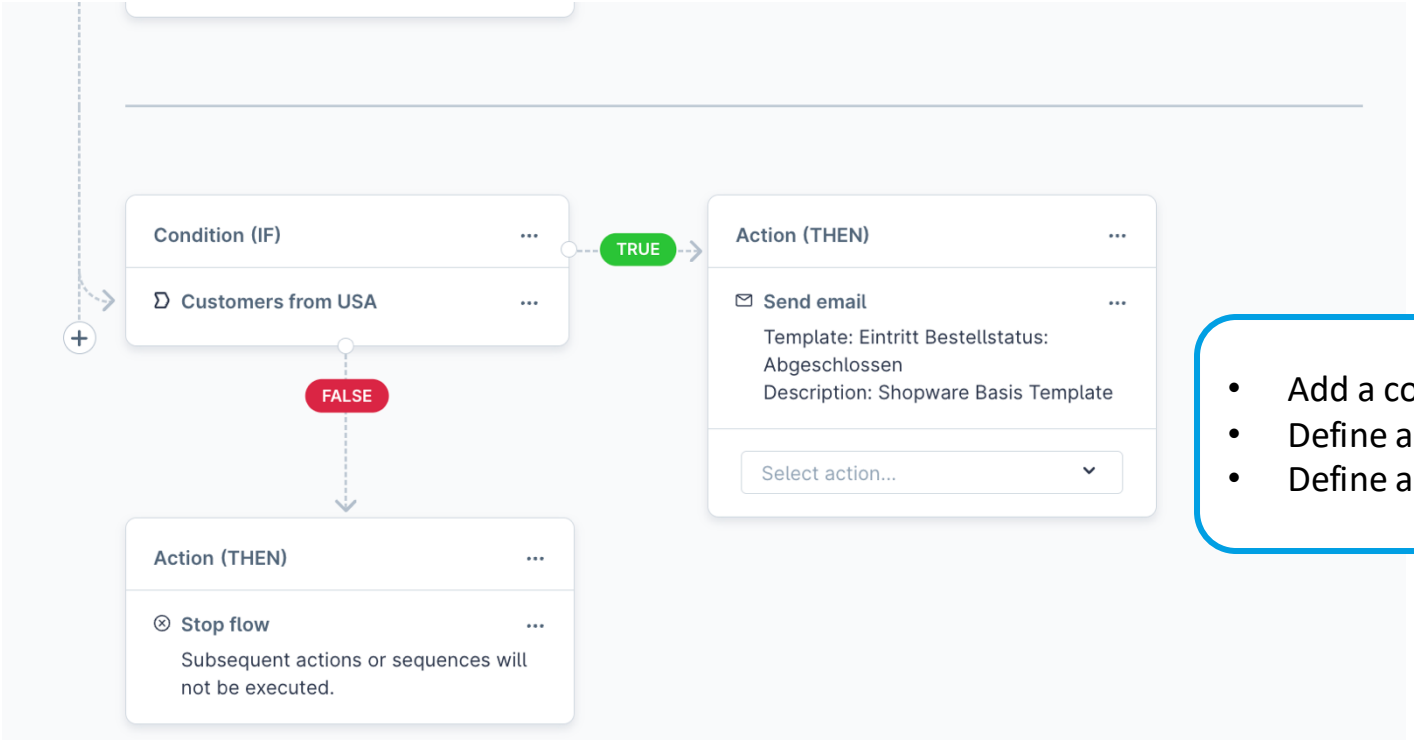
- Add a new flow
- Define general information:
  name, description, priority, status

Flowbuilder introduction



- Configure the flow
- Select a trigger
- Select an action

Flowbuilder introduction

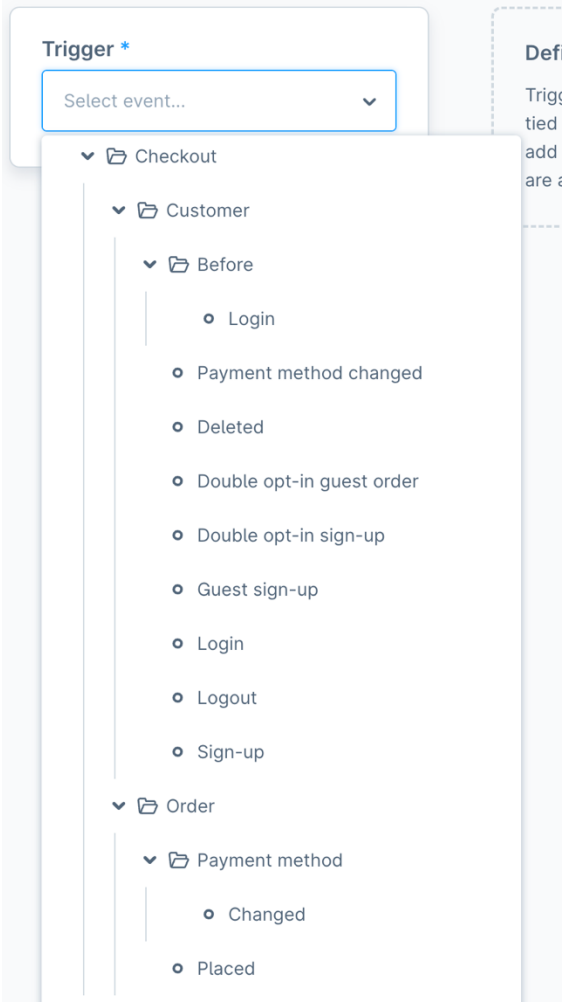


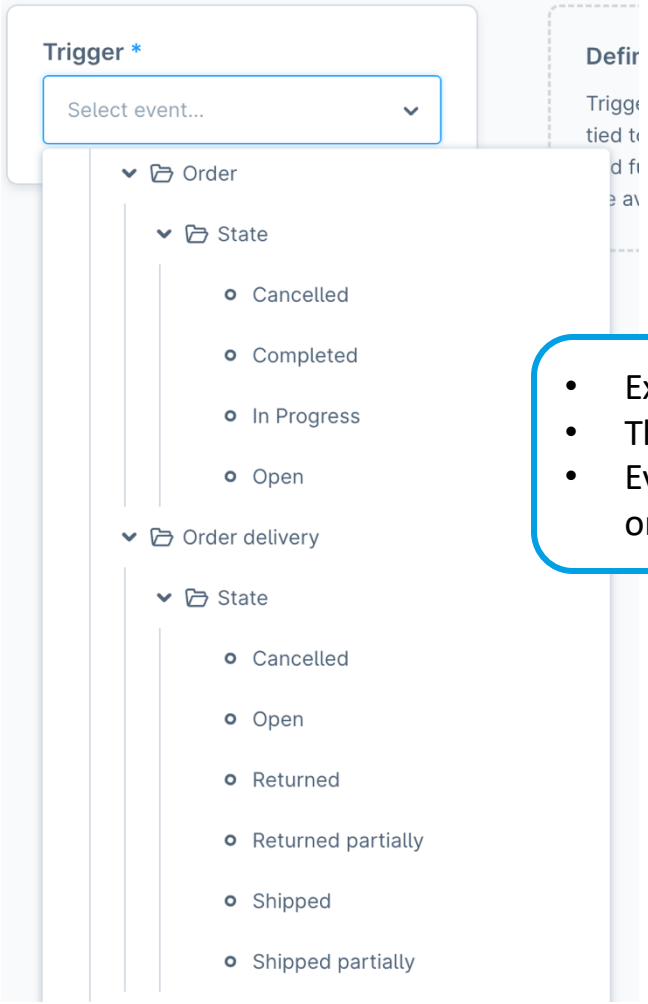
- Add a condition to the trigger
- Define action for false condition
- Define action for true condition

Flowbuilder introduction

- **Flow**: Automation process. Has a trigger and several conditions and trigger.
- **Trigger**: The event that starts a flow (e.g., order placed, payment confirmed).
- **Conditions**: The rules that define when a workflow will run (e.g., if the order total is above a certain value).
- **Actions**: The tasks that are executed when conditions are met (e.g., sending an email, updating stock).

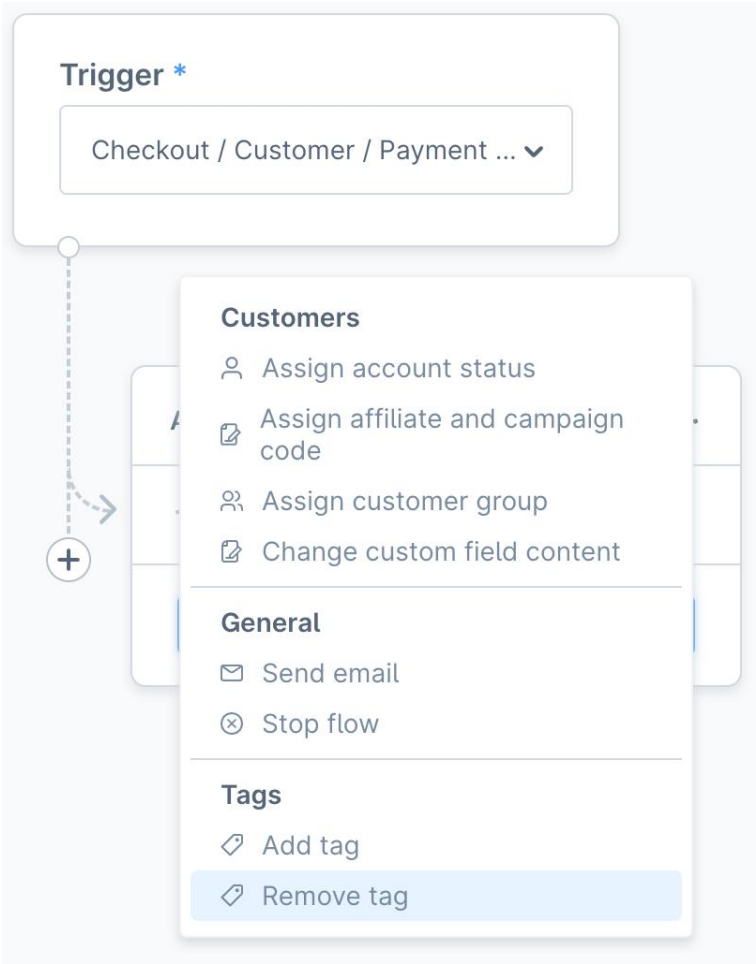
Flowbuilder introduction

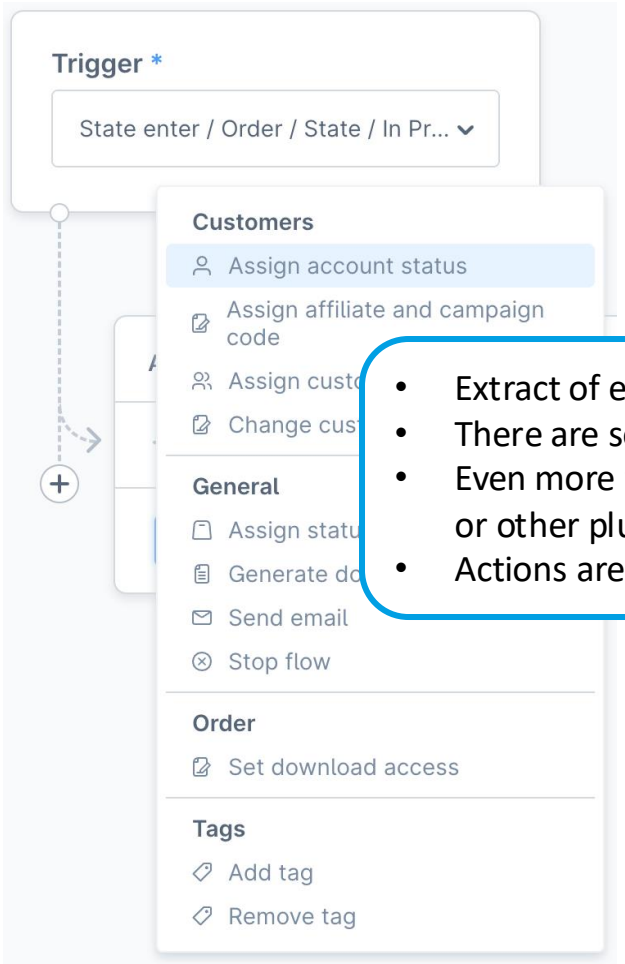




- Extract of existing trigger
- There are some more
- Even more with rise, evolve or beyond license or other plugins

Flowbuilder introduction





- Extract of existing actions
- There are some more
- Even more with rise, evolve or beyond license or other plugins
- Actions are available depending on the trigger

MAKAIRA.io

Example description

- We want to implement a "product back in stock" feature
- If the stock changes from 0 to something greater than 0, customers on the subscriber list should receive an e-mail notification
- If customers have added themselves to a subscriber list, they should receive a voucher code that is only valid for customers on the subscriber list

Implementation the classic way

- Create plugin or app
- Create listener for stock changes
- Send an e-mail when the stock changes from 0 to greater than 0
- Send e-mail with voucher code when the customer subscribed to the subscriber list
- Implementation only does exactly what was required

Implementation based on the flowbuilder

- Create flowbuilder trigger for product stock changes
- Create flowbuilder action to set custom field values for products
- Create rule builder rule for product custom fields
- Create flowbuilder trigger for subscriber subscribed and unsubscribed
- With these extensions in the flow and rule builder, the implementation does exactly what was required.
- However, we additional can use the extensions for other features and flows without further programming

Example part 1: add custom trigger for stock changes

- Create custom flow trigger event based on <FlowEventAware>
- Register event for flowbuilder
- Add listener for stock changes and dispatch flow trigger event
- Create flow with custom trigger in shopware administration

Create custom flow trigger event

```php
class ProductStockChangedFlowEvent extends Event implements FlowEventAware, ProductAware, MailAware, LogAware, ScalarValuesAware
{
    public const EVENT_NAME = 'product.changed.stock';

    public function __construct(
        protected Context $context,
        protected string $productId,
        protected int $stockBefore,
        protected int $stockAfter,
        protected Level $logLevel,
        protected MailRecipientStruct $recipients,
    ) {
    }

    /* required by FlowEventAware */
    public function getName(): string
    {
        return self::EVENT_NAME;
    }

    /* required by FlowEventAware */
    public function getContext(): Context
    {
        return $this->context;
    }
```

- **FlowEventAware:** required for all flowbuilder trigger
- **ProductAware:** make product data and actions available
- **MailAware:** activate mail actions
- **LogAware:** log data to the event logs
- **ScalarValueAware:** to use scalar value in the event

Create custom flow trigger event

```php
/* required by ProductAware */
public function getProductId(): string
{
    return $this->productId;
}

/* required by MailAware */
public function getSalesChannelId(): ?string
{
    if (method_exists($this->context->getSource(), method: 'getSalesChannelId')) {
        return $this->context->getSource()->getSalesChannelId();
    }

    return null;
}

/* required by MailAware */
public function getMailStruct(): MailRecipientStruct
{
    return $this->recipients;
}
```

Create custom flow trigger event

```php
/* required by LogAware */
public function getLogData(): array
{
    return [
        'stockBefore' => $this->stockBefore,
        'stockAfter' => $this->stockAfter,
    ];
}

/* required by LogAware */
public function getLogLevel(): Level
{
    return $this->logLevel;
}
```

Create custom flow trigger event

```php
    public static function getAvailableData(): EventDataCollection
    {
        return (new EventDataCollection())
            ->add( name: 'product', new EntityType( definitionClass: ProductDefinition::class))
            ->add( name: 'stockBefore', new ScalarValueType( type: 'int'))
            ->add( name: 'stockAfter', new ScalarValueType( type: 'int'));
    }

    public function getValues(): array
    {
        return [
            'stockBefore' => $this->stockBefore,
            'stockAfter' => $this->stockAfter,
        ];
    }
}
```

Register event for flowbuilder

```php
class ProductStockChangedListener implements EventSubscriberInterface
{
    public function __construct(
        protected BusinessEventCollector $businessEventCollector,
        protected EventDispatcherInterface $eventDispatcher,
        protected ProductService $productService,
        protected StockSubscriberService $stockSubscriberService,
    ) {
    }

    public static function getSubscribedEvents()
    {
        return [
            PostWriteValidationEvent::class => 'onPostWriteValidation',
            PreWriteValidationEvent::class => 'onPreWriteValidation',
            BusinessEventCollectorEvent::NAME => ['onBusinessEventCollect', 1000],
        ];
    }
```

- Listen to <BusinessEventCollectorEvent> to register the flow trigger event

Register event for flowbuilder

```php
public function onBusinessEventCollect(BusinessEventCollectorEvent $event): void
{
    $collection = $event->getCollection();

    $definition = $this->businessEventCollector->define( class: ProductStockChangedFlowEvent::class);

    if (!$definition) {
        return;
    }

    $collection->set($definition->getName(), $definition);
}
```

Add listener for stock changes / dispatch event

```php
class ProductStockChangedListener implements EventSubscriberInterface
{
    public function __construct(
        protected BusinessEventCollector $businessEventCollector,
        protected EventDispatcherInterface $eventDispatcher,
        protected ProductService $productService,
        protected StockSubscriberService $stockSubscriberService,
    ) {
    }

    public static function getSubscribedEvents()
    {
        return [
            PostWriteValidationEvent::class => 'onPostWriteValidation',
            PreWriteValidationEvent::class => 'onPreWriteValidation',
            BusinessEventCollectorEvent::NAME => ['onBusinessEventCollect', 1000],
        ];
    }
```

- Listen to <PreWriteValidationEvent> and <PostValidationEvent> for product changes close to the database

Add listener for stock changes / dispatch event

```php
public function onPreWriteValidation(PreWriteValidationEvent $event): void
{
    foreach ($event->getCommands() as $command) {
        if ($command->getEntityName() === ProductDefinition::ENTITY_NAME
            && $command instanceof UpdateCommand
        ) {
            if ($command->hasField( storageName: 'stock')) {
                $command->requestChangeSet();
            }
        }
    }
}
```

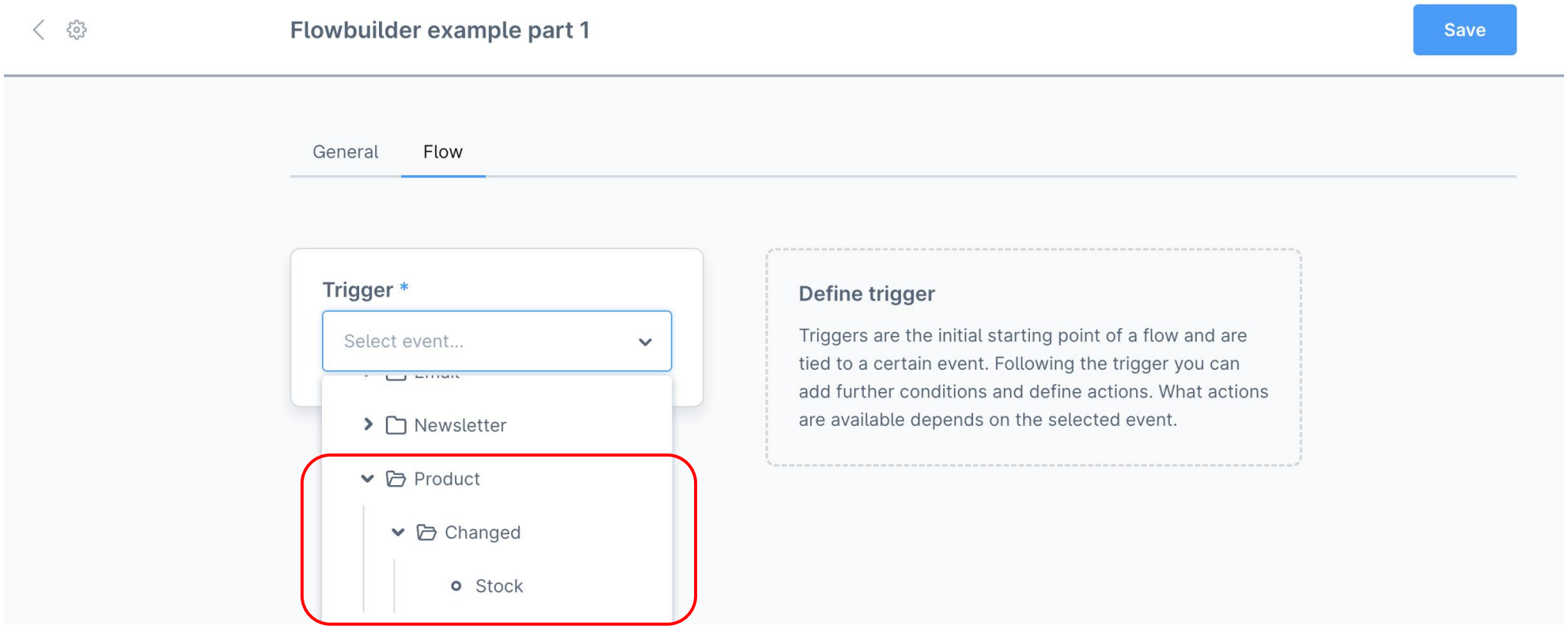
- Request change set if stock value changed

Add listener for stock changes / dispatch event

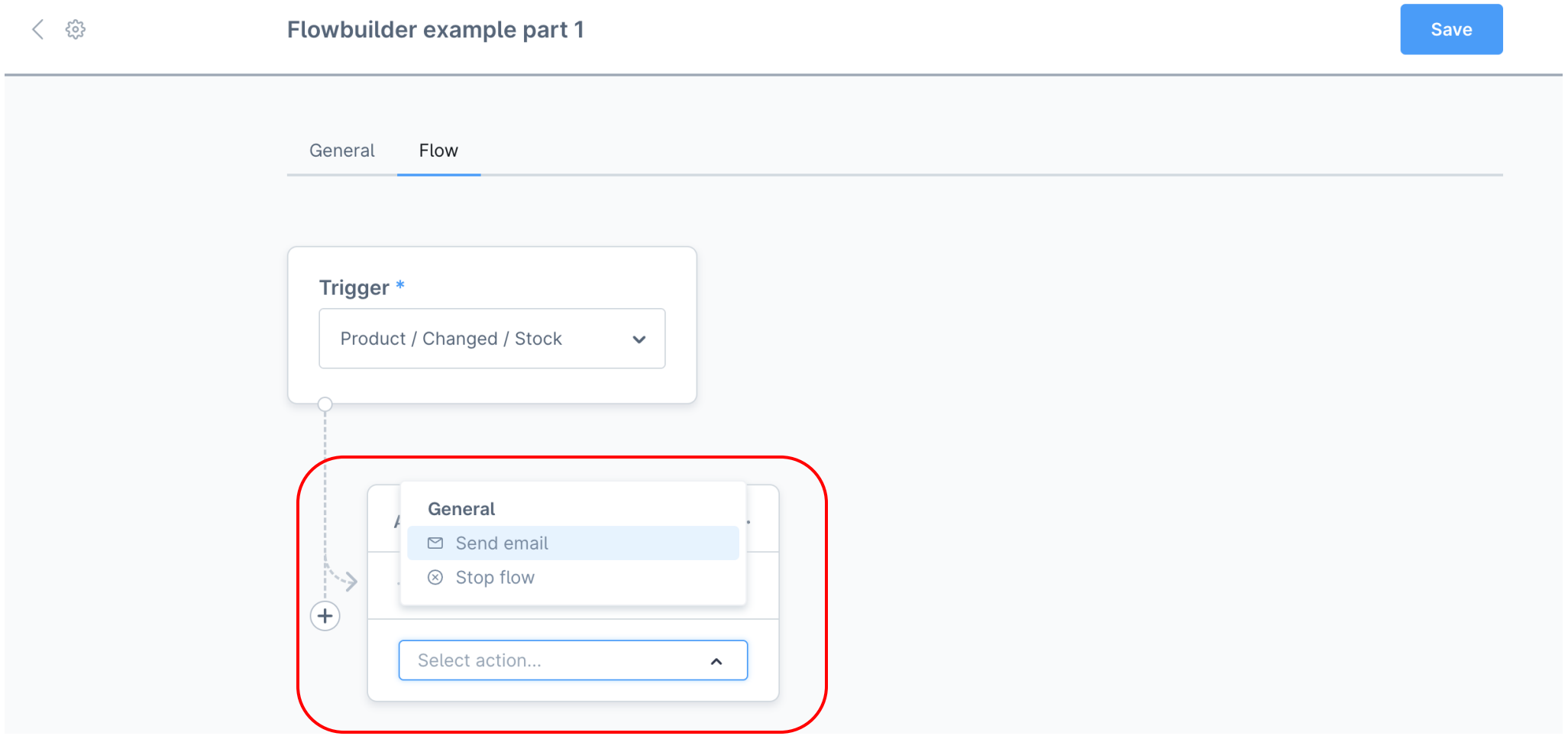
```php
public function onPostWriteValidation(PostWriteValidationEvent $event): void
{
    foreach ($event->getCommands() as $command) {
        if ($command->getEntityName() === ProductDefinition::ENTITY_NAME
            && $command instanceof UpdateCommand
        ) {
            if ($command->getChangeSet() === null) {
                continue;
            }

            $productId = $command->getDecodedPrimaryKey()['id'];

            $productStockChangedEvent = new ProductStockChangedFlowEvent(
                $event->getContext(),
                $productId,
                (int) $command->getChangeSet()->getBefore( property: 'stock'),
                (int) $command->getChangeSet()->getAfter( property: 'stock'),
                logLevel: Level::Info,
                new MailRecipientStruct([])
            );
            $this->eventDispatcher->dispatch($productStockChangedEvent, eventName: ProductStockChangedFlowEvent::EVENT_NAME);
        }
    }
}
```

- Dispatch flow trigger event for product stock changed

Create flow with custom trigger



Create flow with custom trigger



Example part 2: add custom field action for product entity

- Create flow action based on <FlowAction>
- Add action to flow sequence component
- Create flow with custom action in shopware administration

Create flow action

```php
class SetProductCustomFieldAction extends FlowAction
{
    use CustomFieldActionTrait;

    public function __construct(
        protected Connection $connection,
        protected EntityRepository $productRepository,
        protected StringTemplateRenderer $templateRenderer
    ) {
    }

    public static function getName(): string
    {
        return 'action.set.product.custom.field';
    }

    public function requirements(): array
    {
        // return []; => available for all flow trigger
        return [ProductAware::class]; // => available for specific flow trigger
    }
```

- Create action based on <FlowAction>
- Use existing <CustomFieldActionTrait>
- Define requirements

Create flow action

```php
public function handleFlow(StorableFlow $flow): void
{
    if (!$flow->hasData( key: ProductAware::PRODUCT)) {
        return;
    }

    /** @var ProductEntity $product */
    $product = $flow->getData( key: ProductAware::PRODUCT);

    $customFields = $this->getCustomFieldForUpdating($product->getCustomfields(), $flow->getConfig());

    if ($customFields === null) {
        return;
    }

    $customFieldName = $this->getCustomFieldNameFromId($flow->getConfig()['customFieldId'], $flow->getConfig()['entity']);
    $customField = $customFields[$customFieldName] ?? null;
    if ($customField === null) {
        return;
    }
```

- Add action handler
- Handle custom field values

Create flow action

```
        $renderedValue = $this->templateRenderer->render($customField, $flow->getVars()['data'], $flow->getContext());
        $renderedCustomFields[$customFieldName] = $renderedValue;
        $customFields[$customFieldName] = $renderedValue;

        $product->setCustomfields($customFields);
        $flow->setData('product', $product);

        $this->productRepository->upsert([
            [
                'id' => $product->getId(),
                'customFields' => $renderedCustomFields,
            ],
        ], $flow->getContext());
    }
}
```

- Parse custom field value with template renderer to use variables in flow action configuration
- Save custom fields to product

Create flow action

```xml
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <!-- Actions -->
        <service id="ShopwareFlowBuilderStockExample\Content\Product\Action\SetProductCustomFieldAction">
            <argument type="service" id="Doctrine\DBAL\Connection"/> Connection
            <argument type="service" id="product.repository"/> EntityRepository
            <argument type="service" id="Shopware\Core\Framework\Adapter\Twig\StringTemplateRenderer"/> StringTemplateRenderer
            <tag name="flow.action" priority="350" key="action.set.product.custom.field"/>
        </service>
```

Add action to flow sequence component

```
Component.override('sw-flow-sequence-action', {
    inject: [
        'flowBuilderService',
    ],

    computed: {
        groups() { // only required if new group
            this.actionGroups.unshift(GROUP);

            return this.$super('groups');
        },

        modalName() {
            if (this.selectedAction === ACTION.SET_PRODUCT_CUSTOM_FIELD) {
                return 'sw-flow-set-entity-custom-field-modal';
            }

            return this.$super('modalName');
        },
    },
```

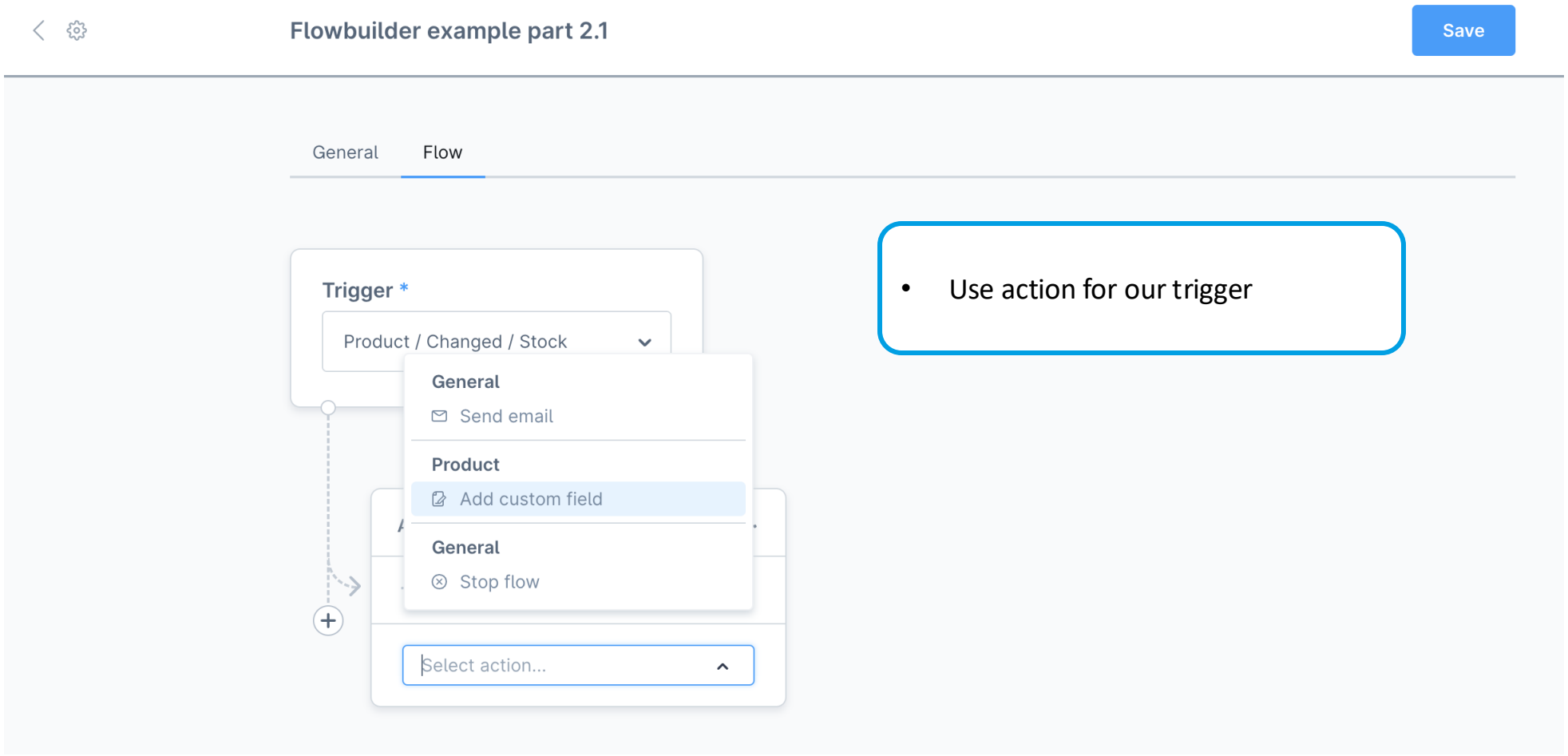
- Add action group to flow sequence component, because we have a new group
- Define action configuration modal component
- We can use the existing custom field modal component

Add action to flow sequence component

```
    created() {
        this.flowBuilderService.$entityAction[ACTION.SET_PRODUCT_CUSTOM_FIELD] = 'product';
    },

    methods: {
        getActionTitle(actionName) {
            if (actionName === ACTION.SET_PRODUCT_CUSTOM_FIELD) {
                return {
                    value: actionName,
                    icon: 'regular-file-signature',
                    label: this.$tc('set-product-custom-field-action.title'),
                    group: GROUP,
                }
            }

            return this.$super('getActionTitle', actionName);
        },
    },
});
```

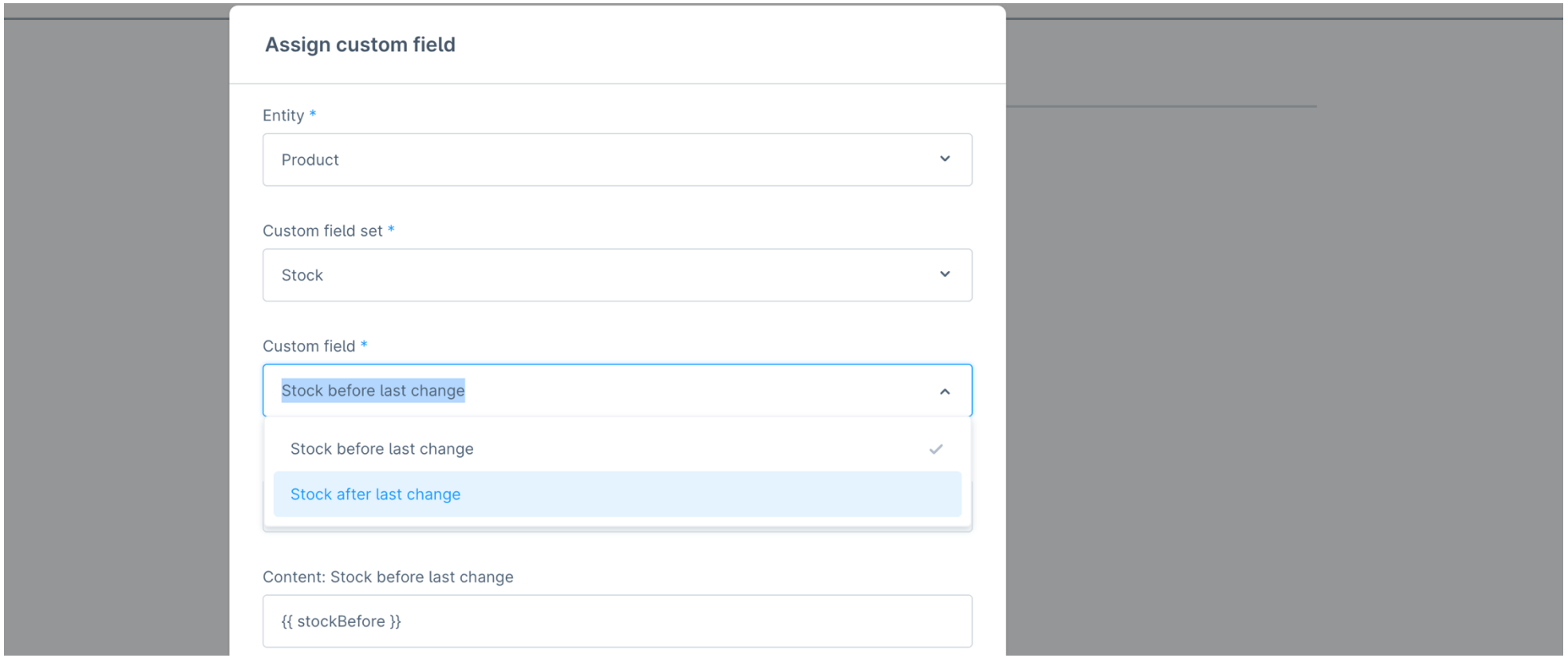
- Add action to flow builder service
- Add action title to flow sequence component

Create flow with custom action

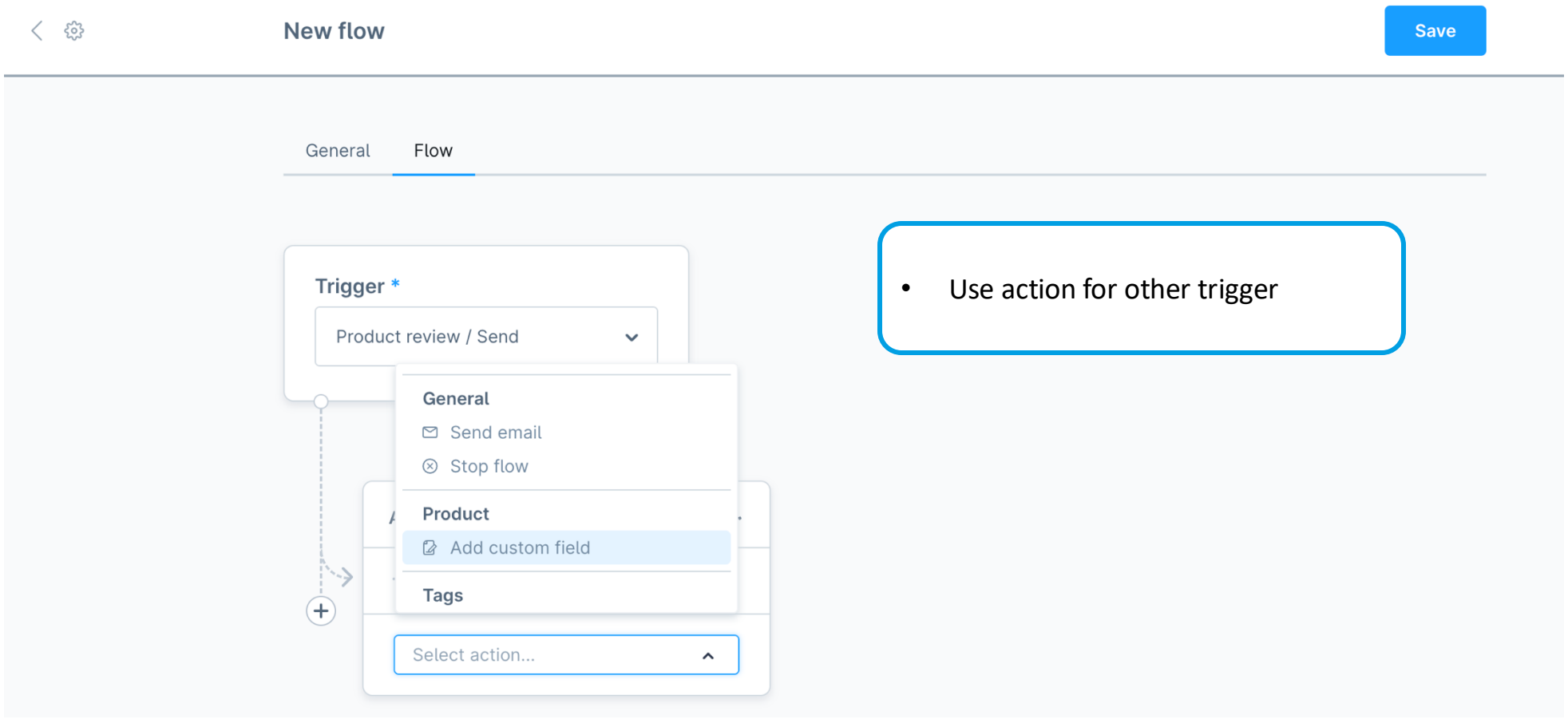


- Use action for our trigger

Create flow with custom action



Create flow with custom action



- Use action for other trigger

Example part 3: add custom rule for product custom fields

- Add product custom field rule to rule builder
- Add rule scope
- Register rule and group
- Add rule condition type component
- Add flow with custom rule in administration

Add product custom field rule to rule builder

```php
class ProductCustomFieldRule extends Rule
{
    final public const RULE_NAME = 'productCustomField';

    protected array|string|int|bool|float|null $renderedFieldValue = null;

    protected ?string $selectedField = null;

    protected ?string $selectedFieldSet = null;

    public function __construct(
        protected string $operator = self::OPERATOR_EQ,
        protected array $renderedField = [],
    ) {
        parent::__construct();
    }
```

- Add custom rule based on <Rule>

Add product custom field rule to rule builder

```php
public function match(RuleScope $scope): bool
{
    if (!$scope instanceof ProductCustomFieldScope) {
        return false;
    }

    $productCustomFields = $scope->getProduct()->getCustomFields() ?? [];

    return CustomFieldRule::match($this->renderedField, $this->renderedFieldValue, $this->operator, $productCustomFields);
}

public function getConstraints(): array
{
    return CustomFieldRule::getConstraints($this->renderedField);
}
}
```

- Use existing <CustomFieldRule> for condition match

```xml
<!-- Rules -->
<service id="ShopwareFlowBuilderStockExample\Content\Product\Rule\ProductCustomFieldRule">
    <tag name="shopware.rule.definition" />
</service>
```

Add rule scope

```php
class ProductCustomFieldScope extends RuleScope
{
    public function __construct(
        protected ProductEntity $product,
        protected SalesChannelContext $salesChannelContext,
    ) {
    }

    public function getProduct(): ProductEntity
    {
        return $this->product;
    }

    public function getContext(): Context
    {
        return $this->salesChannelContext->getContext();
    }

    public function getSalesChannelContext(): SalesChannelContext
    {
        return $this->salesChannelContext;
    }
}
```

- Add rule scope based on <RuleScope>

Register rule and group

```
Shopware.Application.addServiceProviderDecorator('ruleConditionDataProviderService', (ruleConditionService) => {

    ruleConditionService.upsertGroup('product', {
        id: 'product',
        name: 'sw-settings-rule.detail.groups.product',
    });

    ruleConditionService.addCondition('productCustomField', {
        component: 'sw-condition-product-custom-field',
        label: 'sw-condition.condition.product.productCustomFieldRule',
        scopes: ['product'],
        group: 'product',
    });

    return ruleConditionService;
});
```

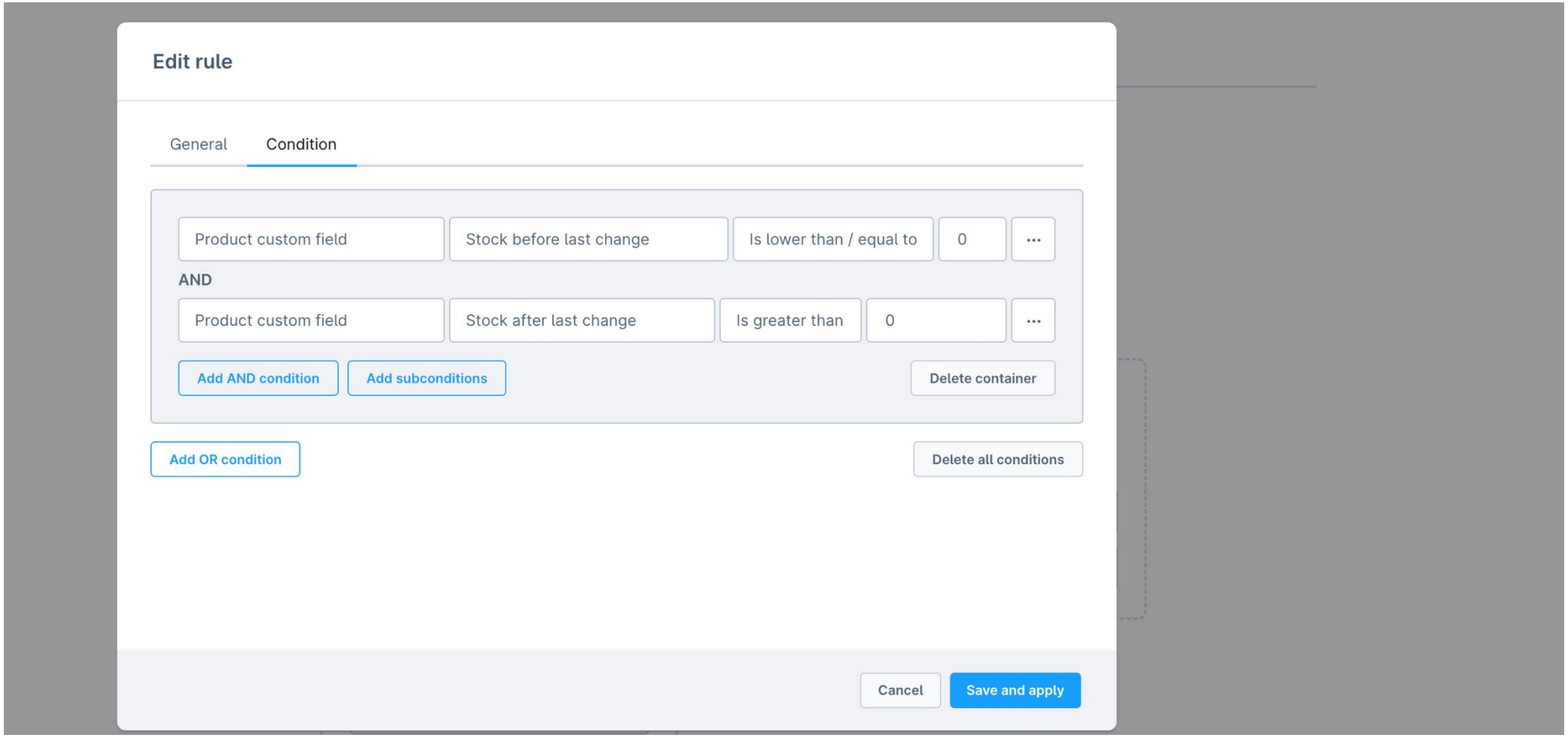
- Add rule and rule group to rule condition data provider service

Add rule condition type component

```
Component.extend('sw-condition-product-custom-field', 'sw-condition-base', {
    template,

    inject: [
        'repositoryFactory',
    ],

    mixins: [
        Mixin.getByName('sw-inline-snippet'),
    ],

    computed: {
        customFieldCriteria() {
            const criteria = new Criteria(1, 25);
            criteria.addAssociation('customFieldSet');
            criteria.addFilter(Criteria.equals('customFieldSet.relations.entityName', 'product'));
            criteria.addSorting(Criteria.sort('customFieldSet.name', 'ASC'));
            return criteria;
        },
```

- Add rule condition type component for product custom fields

Add flow with custom rule



Example part 4: add custom subscriber list for mail action

- (Add entity and api for subscriber list) => not part of the presentation
- Add subscriber list to product stock change trigger
- Extend send mail action component and add subscriber list selection
- Use subscriber list in send mail action

Add subscriber list to product stock change trigger

```php
$productId = $command->getDecodedPrimaryKey()['id'];

$productStockChangedEvent = new ProductStockChangedFlowEvent(
    $event->getContext(),
    $productId,
    (int) $command->getChangeSet()->getBefore( property: 'stock'),
    (int) $command->getChangeSet()->getAfter( property: 'stock'),
     logLevel: Level::Info,
    new MailRecipientStruct($this->getStockSubscriberRecipients($productId, $event->getContext()))
);
$this->eventDispatcher->dispatch($productStockChangedEvent,  eventName: ProductStockChangedFlowEvent::EVENT_NAME);
```

- Extend our existing code in the product stock change listener and add stock subscriber recipients

Add subscriber list to product stock change trigger

```php
protected function getStockSubscriberRecipients(string $productId, Context $context): array
{
    $stockSubscribers = $this->stockSubscriberService->findActiveStockSubscriberForProduct($productId, $context);

    $recipients = [];
    foreach ($stockSubscribers as $stockSubscriber) {
        $recipients[$stockSubscriber->getCustomer()->getEmail()]
            = $stockSubscriber->getCustomer()->getFirstName() . ' ' . $stockSubscriber->getCustomer()->getLastName();
    }
    return $recipients;
}
```

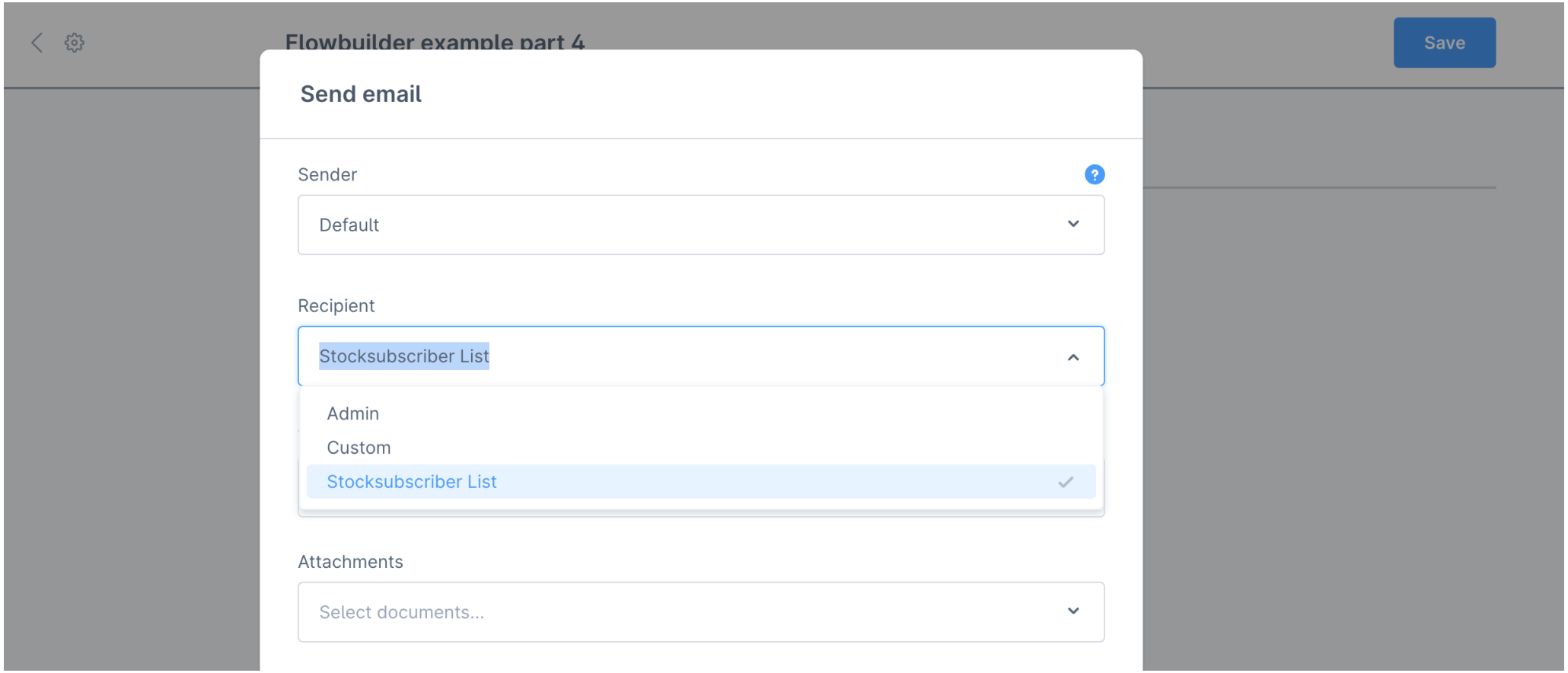
- Load active subscriber for product from the database

Extend send mail action component

```
Component.override('sw-flow-mail-send-modal', {
    computed: {
        recipientOptions() {
            const recipientOptions = this.$super('recipientOptions');

            if (this.triggerEvent.name === 'product.changed.stock') {
                return [
                    ...recipientOptions,
                    {
                        value: 'stockSubscriber',
                        label: 'Stocksubscriber List',
                    }
                ];
            }

            return recipientOptions;
        }
    }
});
```

- Extend send mail action component and add stock subscriber list to the configuration for the product stock changed trigger

Use subscriber list in send mail action



Example part 5: add custom trigger for custom entity with store

- Create custom flow trigger events based on <FlowEventAware>
- Dispatch events in entity api controller
- Add store for subscriber information
- Add flows in shopware administration

Create custom flow trigger events

```php
interface StockSubscriberAware extends FlowEventAware
{
    public const STOCKSUBSCRIBER_ID = 'stockSubscriberId';

    public const STOCKSUBSCRIBER = 'stockSubscriber';

    public function getStockSubscriberId(): string;
}
```

- Add flow event aware interface for stock subscriber

Create custom flow trigger events

```php
class StockSubscriberSubscribedEvent extends Event implements FlowEventAware, MailAware, CustomerAware, StockSubscriberAware
{
    public const EVENT_NAME = 'stock_subscriber.subscribed';

    public function __construct(
        protected Context $context,
        protected MailRecipientStruct $recipients,
        protected string $stockSubscriberId,
        protected string $customerId,
    ) {
    }

    /* required by FlowEventAware */
    public function getName(): string
    {
        return self::EVENT_NAME;
    }

    /* required by FlowEventAware */
    public function getContext(): Context
    {
        return $this->context;
    }
```

- Add flow trigger event for subscriber subscribed
- Use custom <StockSubscriberAware> interface

Create custom flow trigger events

```php
class StockSubscriberUnsubscribedEvent extends Event implements FlowEventAware, MailAware, CustomerAware, StockSubscriberAware
{
    public const EVENT_NAME = 'stock_subscriber.unsubscribed';

    public function __construct(
        protected Context $context,
        protected MailRecipientStruct $recipients,
        protected string $stockSubscriberId,
        protected string $customerId,
    ) {
    }

    /* required by FlowEventAware */
    public function getName(): string
    {
        return self::EVENT_NAME;
    }
```

- Same for subscriber unsubscribed

Dispatch events in entity api controller

```php
if ($stockSubscriber) {
    $stockSubscriberSubscribeEvent = new StockSubscriberSubscribedEvent(
        $salesChannelContext->getContext(),
        new MailRecipientStruct([
            $stockSubscriber->getCustomer()->getEmail()
                => $stockSubscriber->getCustomer()->getFirstName() . ' ' . $stockSubscriber->getCustomer()->getLastName(),
        ]),
        $stockSubscriber->getId(),
        $stockSubscriber->getCustomer()->getId(),
    );
    $this->eventDispatcher->dispatch($stockSubscriberSubscribeEvent, eventName: StockSubscriberSubscribedEvent::EVENT_NAME);
}
```

Dispatch events in entity api controller

```php
if ($stockSubscriber) {
    $stockSubscriberUnsubscribeEvent = new StockSubscriberUnsubscribedEvent(
        $salesChannelContext->getContext(),
        new MailRecipientStruct([
            $stockSubscriber->getCustomer()->getEmail()
                => $stockSubscriber->getCustomer()->getFirstName() . ' ' . $stockSubscriber->getCustomer()->getLastName(),
        ]),
        $stockSubscriber->getId(),
        $stockSubscriber->getCustomer()->getId(),
    );
    $this->eventDispatcher->dispatch($stockSubscriberUnsubscribeEvent, eventName: StockSubscriberUnsubscribedEvent::EVENT_NAME);
}
```

Add store for subscriber information

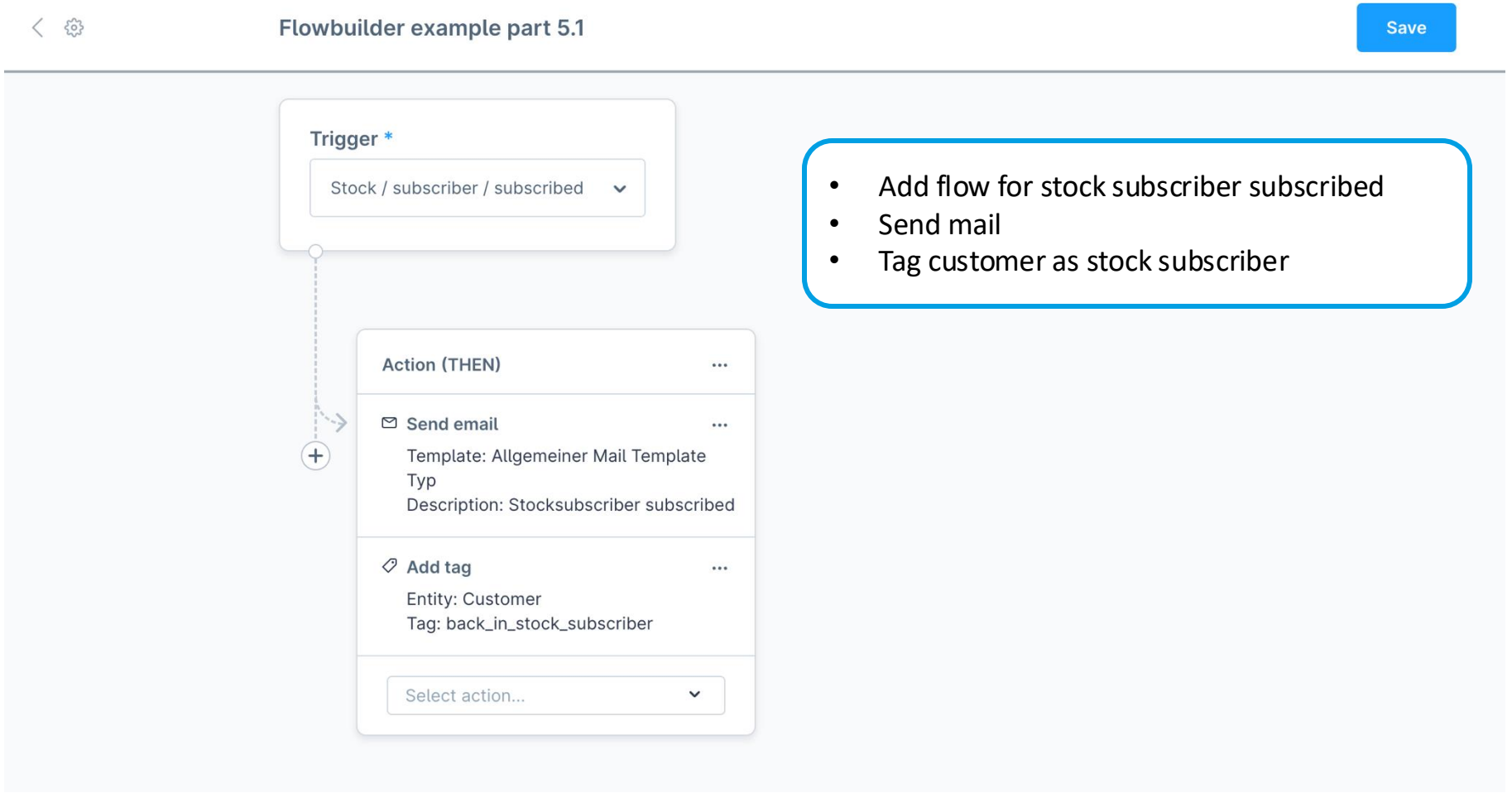
```php
class StockSubscriberStorer extends FlowStorer
{
    public function __construct(protected StockSubscriberService $stockSubscriberService)
    {
    }

    public function store(FlowEventAware $event, array $stored): array
    {
        if (!$event instanceof StockSubscriberAware || isset($stored[StockSubscriberAware::STOCKSUBSCRIBER_ID])) {
            return $stored;
        }

        $stored[StockSubscriberAware::STOCKSUBSCRIBER_ID] = $event->getStockSubscriberId();

        return $stored;
    }
```

- Add stock subscriber store based on <FlowStorer>

```xml
<!-- Storer -->
<service id="ShopwareFlowBuilderStockExample\Content\StockSubscriber\Event\StockSubscriberStorer">
    <argument type="service" id="ShopwareFlowBuilderStockExample\Content\StockSubscriber\StockSubscriberService"/> StockSubscriberService
    <tag name="flow.storer"/>
</service>
```

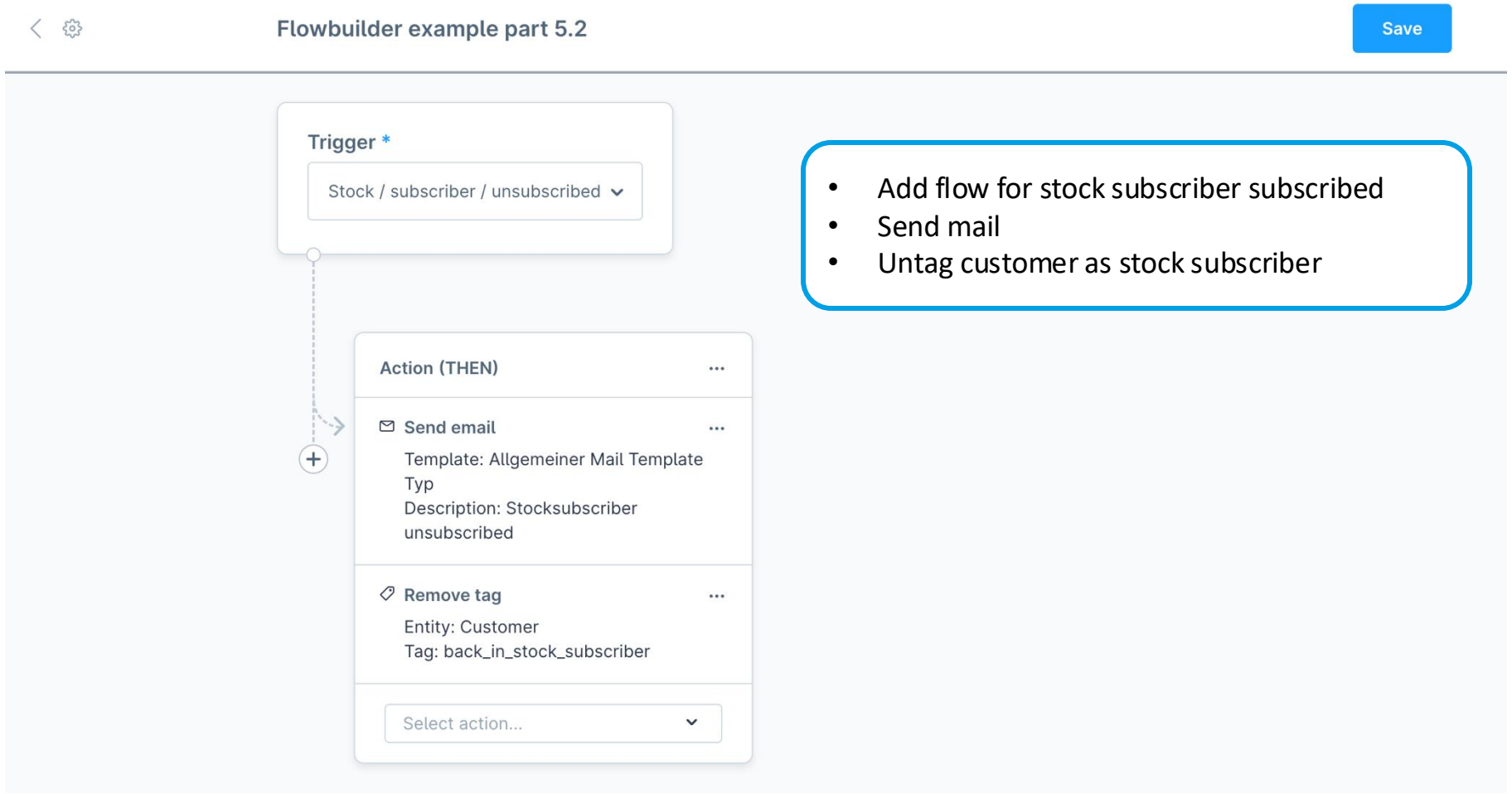
MAKAIRA.io

Add flows in shopware administration



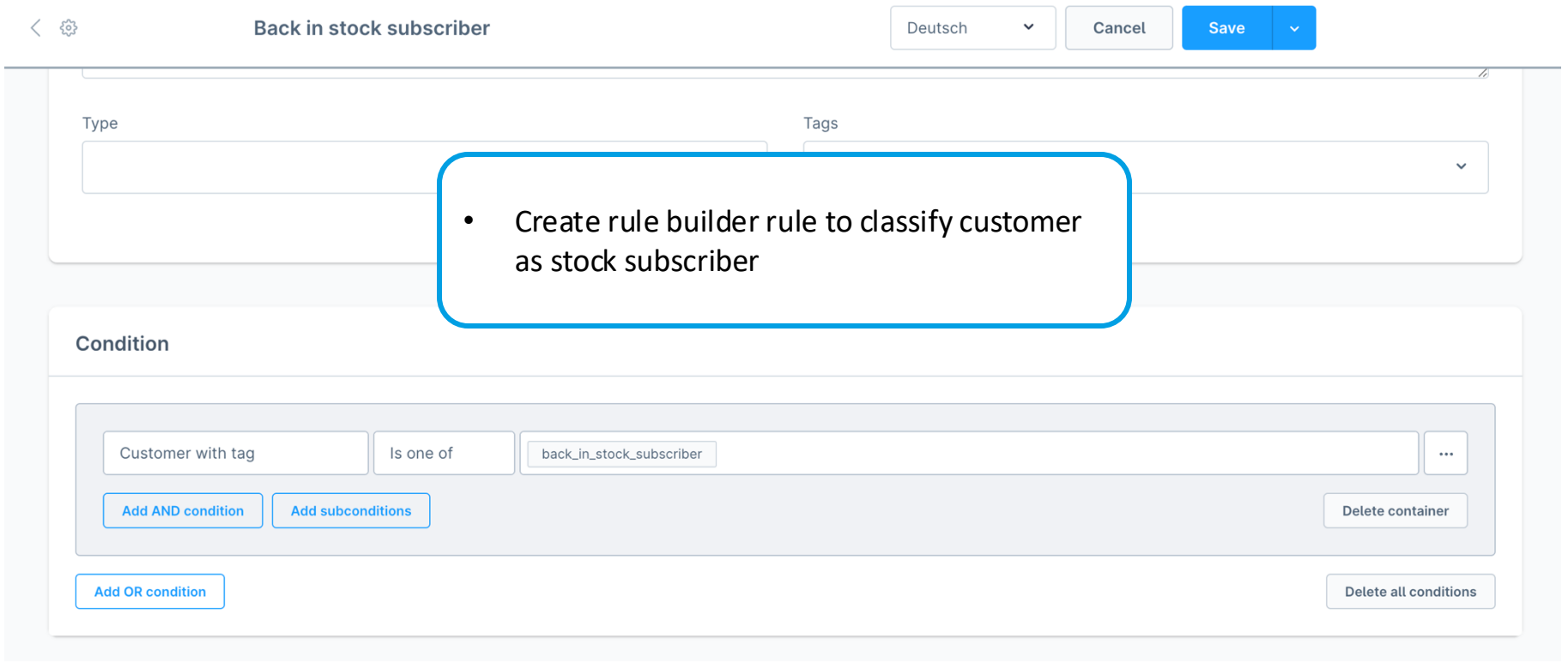
- Add flow for stock subscriber subscribed
- Send mail
- Tag customer as stock subscriber

Add flows in shopware administration



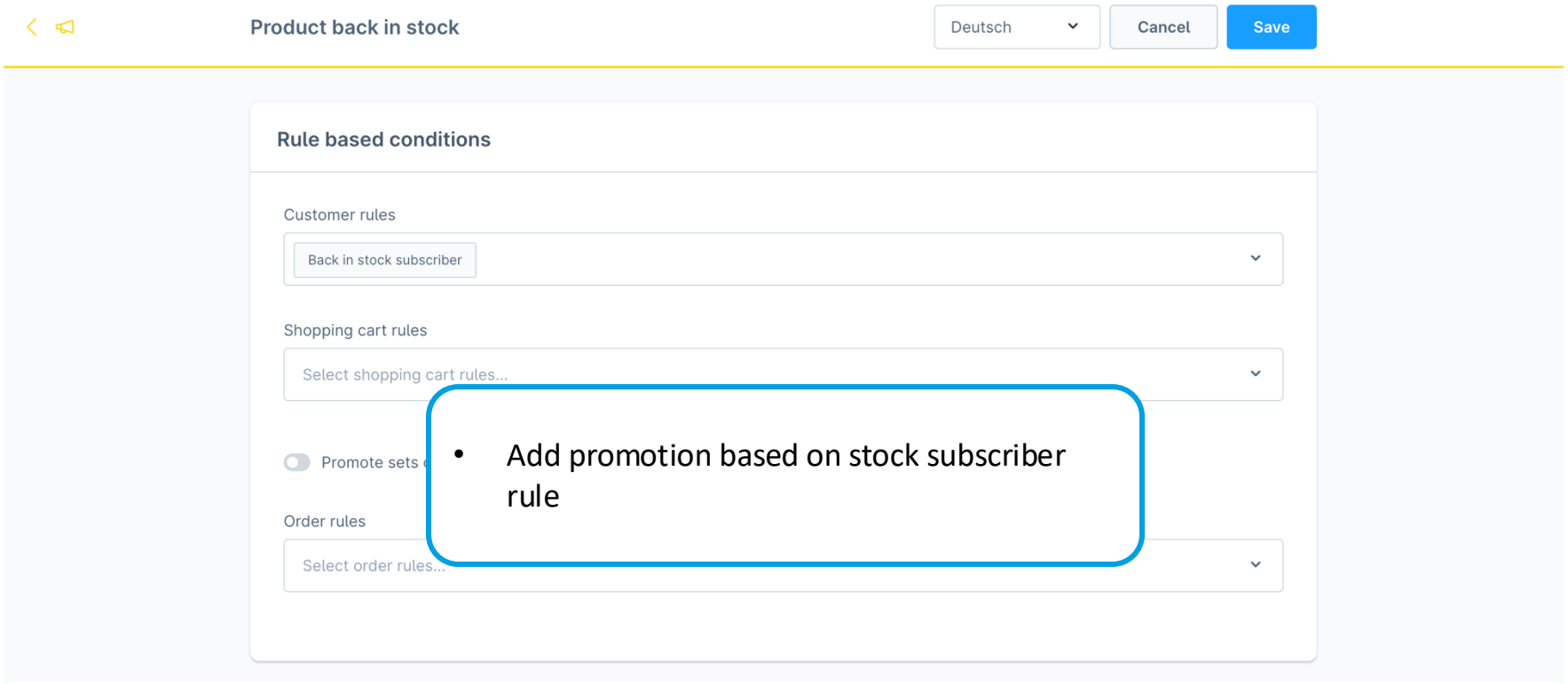
- Add flow for stock subscriber subscribed
- Send mail
- Untag customer as stock subscriber

Add flows in shopware administration



- Create rule builder rule to classify customer as stock subscriber

Add flows in shopware administration

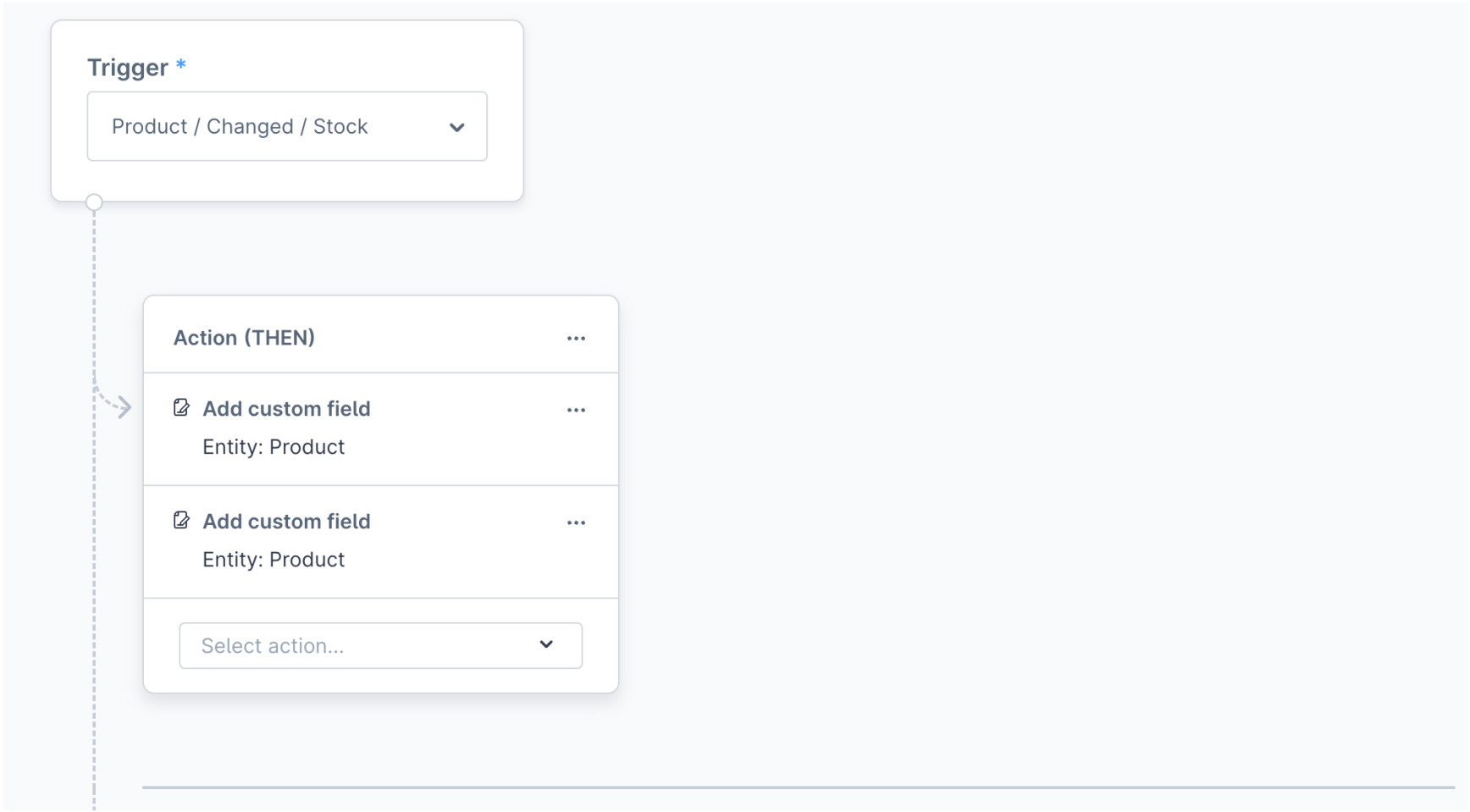


- Add promotion based on stock subscriber rule

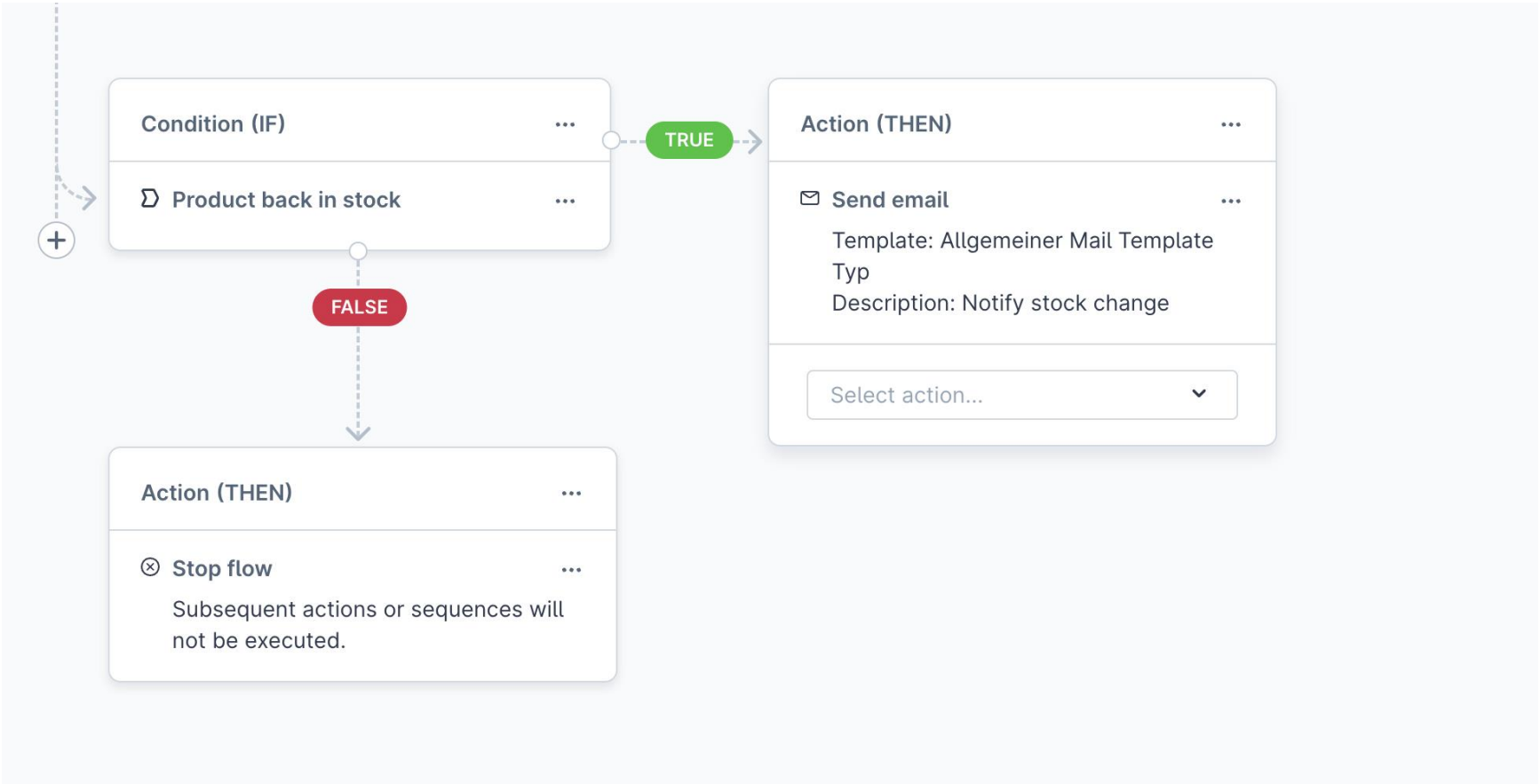
Summary

- We wanted to implement a "product back in stock" feature
- If the stock changes from 0 to something greater than 0, customers on the subscriber list should receive an e-mail notification
- If customers have added themselves to a subscriber list, they should receive a voucher code that is only valid for customers on the subscriber list

Summary


Trigger *
Product / Changed / Stock
Action (THEN)
Add custom field
Entity: Product
Add custom field
Entity: Product
Select action...

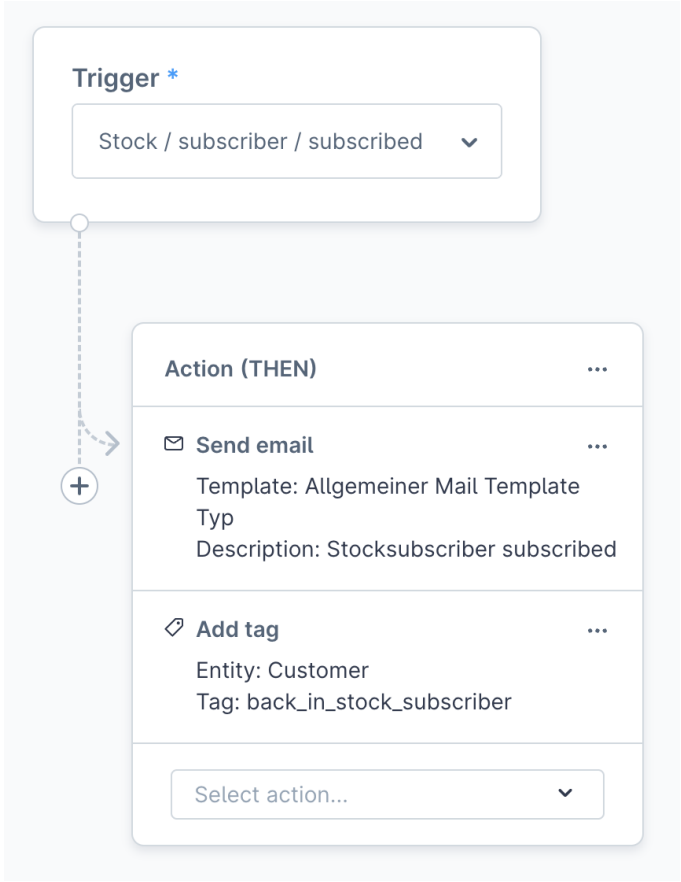
Summary


Condition (IF)
Product back in stock
TRUE
FALSE
Action (THEN)
Send email
Template: Allgemeiner Mail Template Typ
Description: Notify stock change
Select action...
Action (THEN)
Stop flow
Subsequent actions or sequences will not be executed.

Summary

- We wanted to implement a "product back in stock" feature
- **If the stock changes from 0 to something greater than 0, customers on the subscriber list should receive an e-mail notification**
- If customers have added themselves to a subscriber list, they should receive a voucher code that is only valid for customers on the subscriber list

Summary


Trigger *
Stock / subscriber / subscribed
Action (THEN)
Send email
Template: Allgemeiner Mail Template
Typ
Description: Stocksubscriber subscribed
Add tag
Entity: Customer
Tag: back_in_stock_subscriber
Select action...

Summary

- We wanted to implement a "product back in stock" feature
- If the stock changes from 0 to something greater than 0, customers on the subscriber list should receive an e-mail notification
- If customers have added themselves to a subscriber list, they should receive a voucher code that is only valid for customers on the subscriber list

Summary

- It makes it possible to think about features more general in order to reuse individual triggers and events
- The flowbuilder comes with many existing triggers and actions
- And is easy to expand and customize
- The implementation time with or without flowbuilder is nearly the same
- However, implementation with flowbuilder offers more flexibility and reusability for other workflows

Thank you!

Questions?

Contact me:
https://www.linkedin.com/in/mmü

Download code and presentation here

MAKAIRA.io