

Getting the Most Out of Tags

A practical workshop on tag inheritance, event processing, dashboards, alerts and services



Tymon Nowak

Zabbix Summit 2026



What is a tag in Zabbix?

A tag is a name-value pair that adds context.

WITHOUT TAGS

Checkout database is unavailable



WITH TAGS

Checkout database is unavailable

service:checkout
environment:production
component:database

The same tags can then be reused in dashboards, actions, correlation and services.

What tags help us do

Use the same terms in filters, correlation rules and automation.

FIND

Find checkout data without depending on object names.

MATCH

Match a recovery event to the right problem.

ROUTE

Send production alerts to the right recipient.

DISPLAY

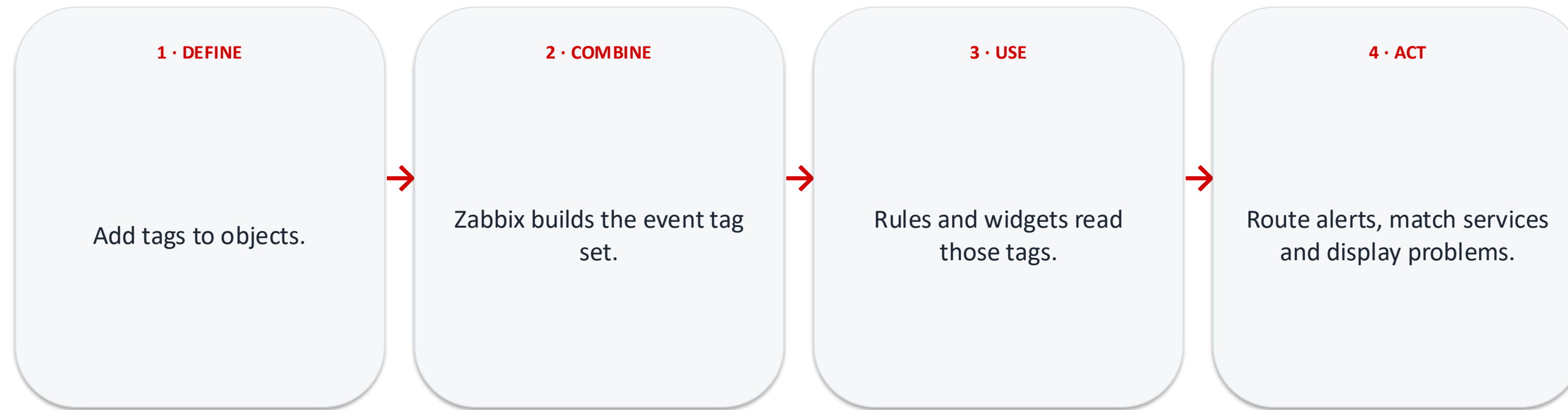
Show only the relevant problems in dashboards and services.

Use the same tag names and values throughout the event lifecycle.



How Zabbix uses tags

A tag starts on an object and becomes part of the event context.



Plan the tag structure first

Use clear values from a controlled list, and never store secrets in tags.

A FEW RULES BEFORE WE START

Use lowercase values.
Keep the vocabulary small and predictable.
One tag = one meaning.
Never put secrets or personal data in tags.

CHECKOUT VOCABULARY

service:checkout
component:frontend | api | database
environment:production | staging
scope:availability | performance
team:payments

Tags work with host groups, applications, inventory and macros. They do not replace them.



Tag names used by Zabbix templates

Official templates reuse a small core vocabulary and add object-specific identity tags.

TEMPLATE CLASSIFICATION

class: network | software | cloud | hardware
subclass: sql | virtualization | containers
target: vendor, product or platform

MONITORED COMPONENT

component: network | system | storage | memory
component: health | interface | cpu | power

EVENT INTENT

scope: availability | performance
scope: notice | capacity | security

DISCOVERED OBJECT IDENTITY

interface, filesystem, database, disk
node, container, pod, namespace

Reuse the official vocabulary where it fits; add business tags such as service and environment consistently.



Agenda

One instructor demo and four practical tasks in the same Checkout environment.

DEMO · INHERITED TAGS

Show the new 8.0 views and tag sources.

01 · COMPLEX PROCESSING

Group related events in an 8.0 CEP window.

02 · TRIGGER CORRELATION

Configure matching recovery by instance.

03 · WIDGETS

Connect tagged items to dashboard widgets.

04 · ALERTS

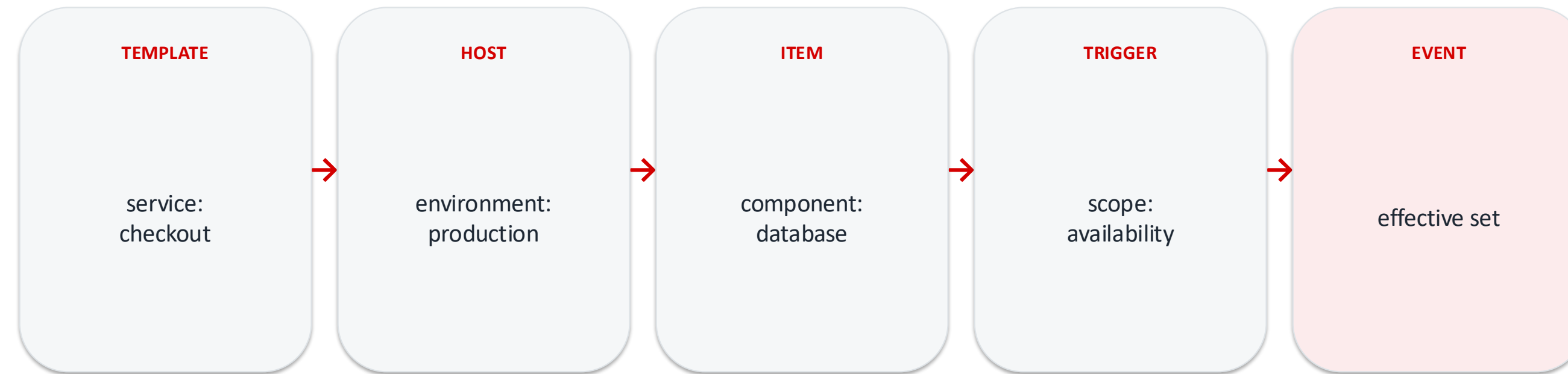
Route alerts and include tag context.



OCTOBER 7 ▪ 9, 2026
RIGA ▪ LATVIA

How event tags are built

Tags from the template, host, item and trigger are added to the problem event.



IMPORTANT: INHERITANCE DOES NOT MEAN OVERRIDE

environment:production + environment:staging

If both reach the event, both values remain. Exact duplicate pairs are stored only once.

Inherited tags are now visible across the interface

The demo follows the same tag from its template source to configuration and monitoring views.

CONFIGURATION FORMS

Templates, hosts, items, triggers and web scenarios can show inherited tags.

OWN OR COMBINED VIEW

Switch between object tags and inherited + object tags.

TAG SOURCE

Outlined icons and Parent templates show where a tag came from.

MONITORING

Latest data displays and filters inherited tags consistently.

This is a visibility change. Event tags still use the final effective tag set.



Import once, then trace every inherited tag

Prepared templates and hosts keep the demo focused on the new 8.0 interface.

1 · TEMPLATES

Import parent and child templates.

2 · HOSTS

Import production and staging hosts.

3 · SOURCES

Compare template and host tags.

4 · OBJECTS

Inspect item, trigger and scenario tags.

5 · MONITORING

Filter inherited tags and open a problem.

Instructor-led demo: participants observe the UI changes instead of building the objects.

Link to download:

[Templates and Hosts](#)



1/5 · Import the workshop templates

Import the parent and child templates with all items and triggers already prepared.

IMPORT FILE

Data collection → Templates → Import

01_tags_workshop_templates.yaml

Contains:

Template Tags Workshop Base

Template Tags Workshop

IMPORT RULES

Update existing: selected

Create new: selected

Delete missing: not selected

Review the comparison, then click Import.

Expected: 2 templates · 4 items · 3 triggers · 1 disabled web scenario



2/5 · Import the workshop hosts

Import two interface-free hosts linked to the prepared child template.

IMPORT FILE

Data collection → Hosts → Import

02_tags_workshop_hosts.yaml

Host group: Tags Workshop
Template: Template Tags Workshop

EXPECTED HOSTS

tags-lab-01
environment:production

tags-lab-02
environment:staging

Interfaces: none

Templates must be imported first because both hosts reference Template Tags Workshop.



3/5 · Compare template and host tag views

Use the own-only and combined views to identify every tag source.

TEMPLATE CHAIN

Template Tags Workshop Base
→ service:checkout
→ team:payments

Template Tags Workshop
→ region:eu-central
→ service:checkout

WHAT TO SHOW

Template tags ↔ Inherited and template tags
Host tags ↔ Inherited and host tags

Outlined source icon
Parent templates
Disabled inherited rows

The repeated service:checkout pair is marked as both local and inherited, then stored once.



4/5 · Inspect item, trigger and scenario tags

Open the imported objects and compare their own tags with the inherited set.

ITEM

Workshop status

Item tags
Inherited and item tags

Own:
component:database

TRIGGER

Checkout database is unavailable

Trigger tags
Inherited and trigger tags

Own:
scope:availability

WEB SCENARIO

Checkout web scenario tag view

Scenario tags
Inherited and scenario tags

Own:
component:frontend

The web scenario is disabled: it exists only to demonstrate the Tags tab.



5/5 · Filter inherited tags and check the event

Use Latest data first, then generate one problem from the imported trigger.

PROBLEM

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.status -o 1
```

RECOVERY

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.status -o 0
```

LATEST DATA CHECK

Filter the production host by:
service equals checkout
team equals payments
region equals eu-central

EXPECTED EVENT TAGS

service:checkout team:payments
region:eu-central environment:production
component:database scope:availability



Two mechanisms used in this workshop

Choose the mechanism from the event lifecycle you need to model.

COMPLEX EVENT PROCESSING

Use a CEP window to group related events and build cause/symptom relationships.

TRIGGER-BASED CORRELATION

Use one trigger with a matching tag when several problems must recover independently.

Task 1 uses CEP. Task 2 uses trigger-based correlation.

Group repeated failures into cause and symptoms

Use the new 8.0.0rc1 CEP window to connect three related problem events.

INPUT

Three database failure events for
instance:payment-01.

CEP RULE

Filter by tags, then group events by
instance in a two-minute window.

RESULT

One cause, two symptoms and
symptom_count:2 on the cause event.

This task uses Data collection → Event processing → Create complex event processing.

1/4 · Inspect the imported CEP source

Confirm that the imported item and trigger can generate repeated problems.

IMPORTED ITEM

Name: Workshop CEP source
Type: Zabbix trapper
Key: workshop.cep
Information type: Numeric (unsigned)

Item tags:
component:database
instance:payment-01

IMPORTED TRIGGER

Name: Checkout database failure for payment-01
Recovery mode: Recovery expression
Problem event generation: Multiple
OK event closes: All problems

Trigger tag: scope:availability

Problem expression

```
last(/Template Tags Workshop/workshop.cep)=1
```

Recovery expression

```
last(/Template Tags Workshop/workshop.cep)=0
```



2/4 · Create the CEP rule and filter

Keep the workshop filter focused on one stable tag.

CREATE THE RULE

Data collection → Event processing
→ Create complex event processing

Name:
Group checkout database events

Type of calculation: And
Sort order: 0
Enabled: yes

CONDITION

Type: Tag value
Tag: instance
Operator: equals
Value: payment-01

Do not add service, environment or component conditions in this lab.

One condition is enough here: instance equals payment-01.



3/4 · Configure the window and operation

Group matching events by instance and mark every event processed by the rule.

TIME WINDOW

Time window: Cause and symptoms grouping
Duration: 2m
Capacity: Unlimited

Group by:
Tag: selected
Tag names: instance

Event count tag:
Yes symptom_count

OPERATION

Operations → Add

Execute when: Event added to window
Operation: Add
Tag: processed_by
Value: cep
Conditions: none

Stop after this rule: selected

Complex event processing

Name

Group checkout database events

Conditions

Label	Name	Actions
A	Tag value component equals database	<a>Edit <a>Remove
<a>Add		

Time window

None

Simple (multiple)

Simple (once)

Cause and symptoms grouping

Pattern match

Duration

2m

Capacity

Unlimited

Limited

Group by

Host group

Host

Tag

instance

tag names

Event count tag

No

Yes

symptom_count

Operations

Details

Conditions

Actions

1: Execute when Event added to window : Add processed_by:cep

Add

Stop after this rule

☒

Sort order

0

Description

Update

Clone

Reset time windows

Delete

Cancel

The first event becomes the cause. Later events in the same instance window become symptoms.



4/4 · Generate and inspect the event group

Send three problem values, inspect the cause and symptoms, then recover them.

3 PROBLEMS

```
for i in 1 2 3; do zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.cep -o 1; sleep 1; done
```

RECOVERY

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.cep -o 0
```

EXPECTED RELATIONSHIP

The first problem is the cause.
The next two problems are symptoms of that cause.

EXPECTED CEP EVIDENCE

Cause tag: symptom_count:2
Event tag: processed_by:cep
History shows the CEP rule action.

Open Monitoring → Problems and expand the cause to inspect its symptoms.



Task completed

Confirm the evidence before moving to the next topic.



PRACTICAL TASK 1 IS COMPLETE

CAUSE

One root problem is visible.

SYMPTOMS

Two related events are attached.

EVIDENCE

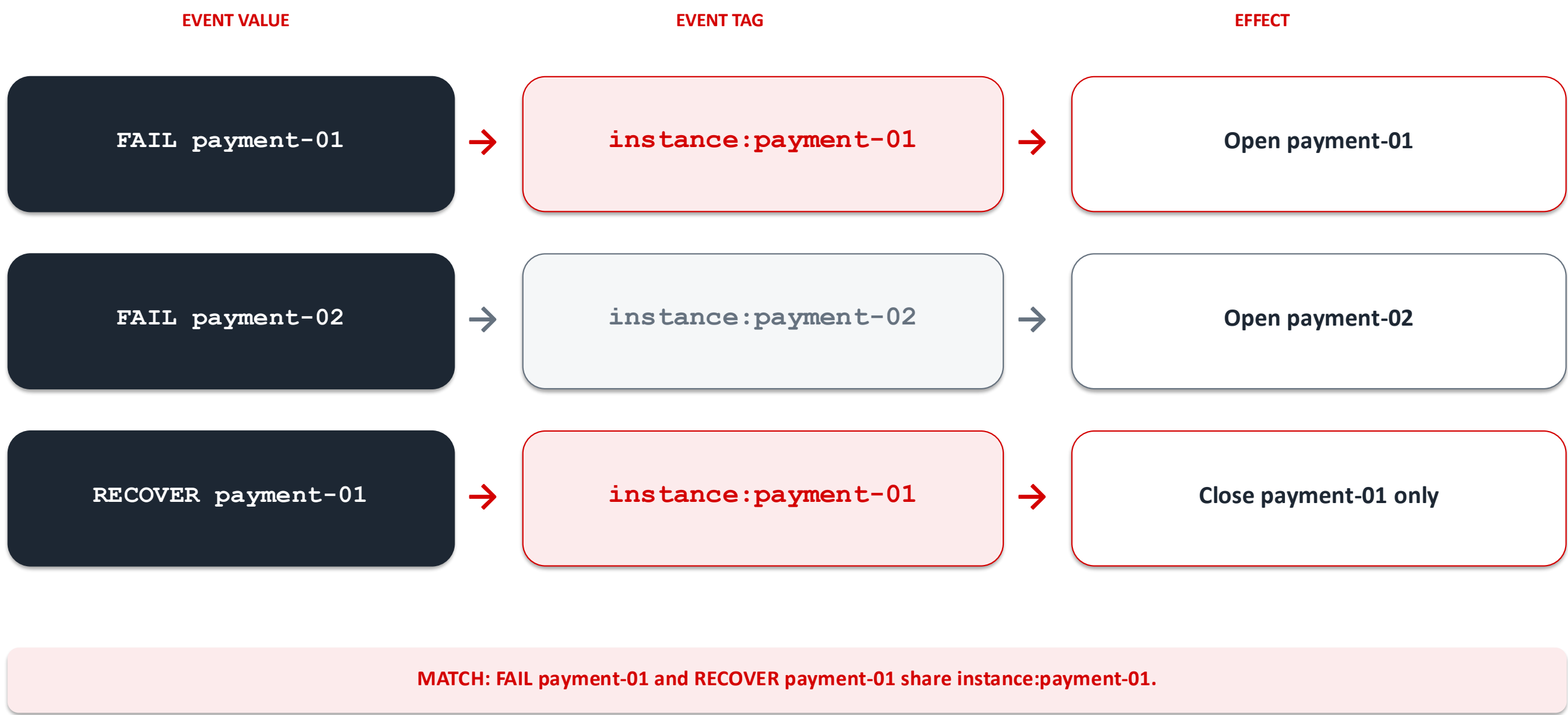
symptom_count:2
processed_by:cep

Next: use a trigger tag to match one recovery event to one open problem.



The tag value connects each event to one problem

The same instance value must be created for the failure and its recovery.



Match recovery events to the right problem

payment-01 and payment-02 fail independently.

THE PROBLEM

When payment-01 recovers, we must close only payment-01.

payment-02 must stay open.

HOW WE SOLVE IT

One text item creates both problems. A dynamic instance tag identifies the problem that each recovery event should close.

We will solve this with a dynamic instance tag.

1/3 · Inspect the imported event stream

Confirm the text contract used by the imported item and trigger.

IMPORTED ITEM

Name: Workshop event stream
Type: Zabbix trapper
Key: workshop.event
Information type: Text

Item tag: component:api

ACCEPTED TEST VALUES

FAIL payment-01
FAIL payment-02
RECOVER payment-01

One item carries every instance.

In production, this data could come from logrt[]. We use a trapper item to keep the lab repeatable.



2/3 · Configure the prepared trigger

Open the imported trigger, change three fields and add one dynamic tag.

OPEN THIS TRIGGER

Data collection → Templates
Template Tags Workshop → Triggers

Checkout component failure: {ITEM.VALUE}

CHANGE THESE THREE SETTINGS

- 1. Problem event generation → Multiple
- 2. OK event closes → All problems if tag values match
- 3. Tag for matching → instance

Add this trigger tag

```
instance: {{ITEM.VALUE}}.regsub ("^ (FAIL|RECOVER) (.+) $" ,\2) }
```

YOUR TRIGGER SETTINGS SHOULD MATCH THIS SCREEN

PROBLEM event generation mode

Single

Multiple

OK event closes

All problems

All problems if tag values match

* Tag for matching

instance



3/3 · Send the test values

Send two failures and one recovery to the same item.

FAIL 01

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.event -o "FAIL payment-01"
```

FAIL 02

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.event -o "FAIL payment-02"
```

RECOVER 01

```
zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.event -o "RECOVER payment-01"
```

EXPECTED RESULT

After the FAIL values, both problems are open.
After RECOVER payment-01, payment-01 is closed and payment-02 remains open.



Task completed

Confirm the evidence before moving to the next topic.



PRACTICAL TASK 2 IS COMPLETE

MATCHING TAG

instance:payment-01

CLOSED

payment-01 is closed.

STILL OPEN

payment-02 remains open.

Next: use item and event tags to build a connected dashboard.



Not every widget filters the same kind of tags

Choose the widget first, then use the tag type it understands.

ITEM NAVIGATOR

Item tags

PROBLEMS

Problem and event tags

HOST

Host tags

SERVICE

Service tags

Graph can use the item selected in Item Navigator.



How tag filters work

Read each operator as a simple question.

EXISTS

Does this tag exist?

EQUALS

Does this tag have this value?

AND

All conditions must match.

OR

Any condition can match.



Build a small dashboard with tags

We will use tags once to find the latency item.

ITEM NAVIGATOR

It finds the latency item by tag and lets us select it.

GRAPH

It reuses the item selected in Item Navigator.

PROBLEMS

It uses its own problem-tag filter.

Item Navigator makes the selection. Graph reuses it.

1/4 · Configure Item Navigator

Filter imported items by their item tag.

DO THIS

Dashboard edit mode → Add widget
Type: Item navigator

Item tags:
component Equals api

Item limit: 100
Save the widget.

Do not send latency data yet.

REFERENCE: ITEM NAVIGATOR SETTINGS

Edit widget

TypeItem navigatorShow header

Namedefault

Refresh intervalDefault (1 minute)

Host groupstype here to searchSelect

Hoststype here to searchSelect

Host tagsAnd/OrOr

tagContainsvalueRemove

Add

Item patternspatternsSelect

Item tagsAnd/OrOr

componentEqualsapiRemove

Add

StateAllNormalNot supported

Show problemsAllUnsuppressedNone

Group byAdd

* Item limit100



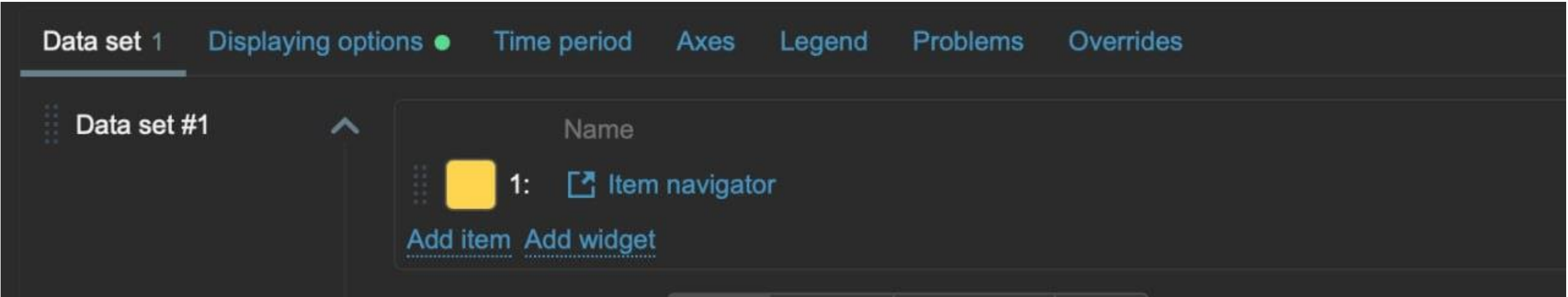
2/4 · Connect Graph to Item Navigator

Use the widget selection instead of a fixed item.

DO THIS

Add widget → Graph | Data set → Item list | Click Add widget | Source: Item navigator
Do not use Add item. Save the Graph widget.

REFERENCE: GRAPH DATA SOURCE



3/4 · Configure the Problems widget

Use problem tags to keep only the production Checkout problem.

DO THIS

Add widget → Problems
Show: Recent problems

Problem tags:
service Contains checkout
environment Equals production

Save the widget.

REFERENCE: PROBLEMS WIDGET SETTINGS

Edit widget

TypeProblemsShow header

Namedefault

Refresh intervalDefault (1 minute)

ShowRecent problemsProblemsHistory

Host groupstype here to searchSelect

Exclude host groupstype here to searchSelect

Hoststype here to searchSelect

Problem

Severity

☐ Not classified☐ Warning☐ High☐ Information☐ Average☐ Disaster

Problem tags

And/OrOr

serviceContainscheckoutRemove

environmentEqualsproductionRemove

4/4 · Send data and verify the dashboard

Send values only after all three widgets are saved.

ITEM USED BY THIS TEST

Checkout latency → checkout.latency | component:api + metric:latency
Item Navigator finds this item; Graph displays its values.

```
SEND VALUES    for v in 120 180 140 220 160; do zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k checkout.latency -o "$v"; sleep 1; done
```

LATEST DATA

Five new latency values are stored.

NAVIGATOR + GRAPH

Select Checkout latency.
Graph shows the series.

PROBLEMS

Production payment-02 remains visible.



Task completed

Confirm the evidence before moving to the next topic.



PRACTICAL TASK 3 IS COMPLETE

ITEM NAVIGATOR

Checkout latency is selectable.

GRAPH

The five values form a series.

PROBLEMS

Production payment-02 is visible.

Next: use event tags to route an alert and include tag context.



How tags help with alerts

Tags answer three practical questions.

SHOULD THIS EVENT TRIGGER THE ACTION?

Action conditions.

WHO SHOULD RECEIVE IT?

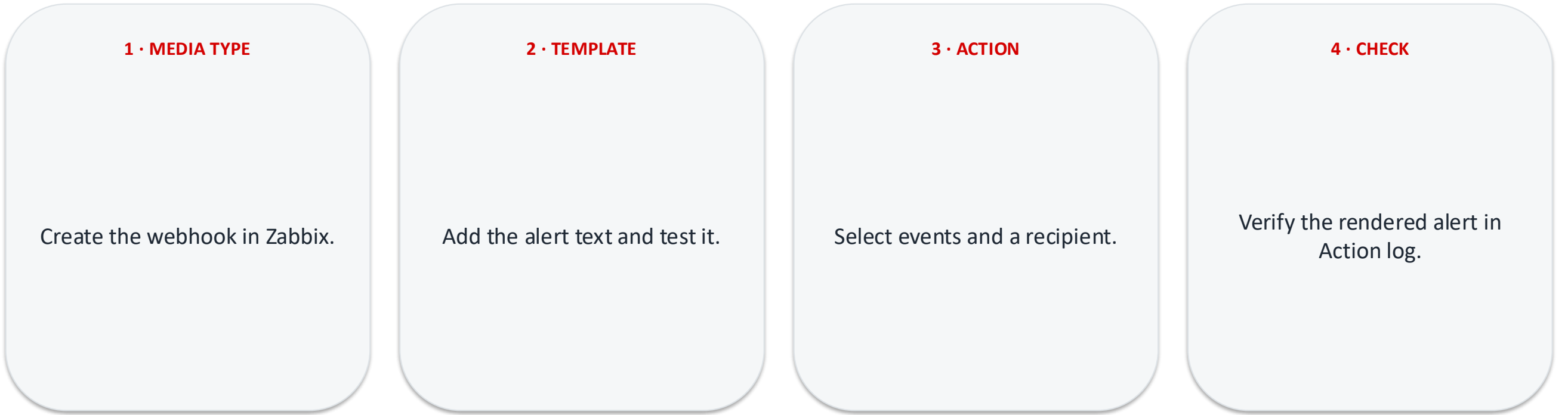
Operation and media type.

WHAT CONTEXT SHOULD WE INCLUDE?

{EVENT.TAGS}
{EVENT.TAGSJSON}

Route imported Checkout events to a local log sink

Build the alerting path around the imported trigger without email or an external endpoint.



The imported trigger supplies the event. You build the media type, message and routing.



1/4 · Create the webhook media type

Paste the JavaScript into the Script field of the webhook media type.

REFERENCE: WEBHOOK MEDIA TYPE

Media type

Message templates 1

Options

* Name

Workshop log sink

Type

Webhook

Parameters

Name	Value	
HTTPProxy		Remove
Message	{ALERT.MESSAGE}	Remove
Subject	{ALERT.SUBJECT}	Remove
To	{ALERT.SENDTO}	Remove
URL		Remove
Add		

* Script

var p = JSON.parse(value);...

* Timeout

30s

Process tags

☐

Webhook script: copy as-is

```
var p = JSON.parse(value);

Zabbix.log(3,
  '[tags-workshop] ' +
  p.Subject + ' | ' + p.Message
);

return 'logged';
```

ZABBIX

SUMMIT²⁶

OCTOBER 7 ▪ 9, 2026

RIGA ▪ LATVIA

2/4 · Add the message template and test

The message template contains the final alert text. The action will use it without an override.

MESSAGE TEMPLATE

Media type → Message templates
→ Add

Message type: Problem
Subject:
Problem: {EVENT.NAME}

Message:
{EVENT.TAGSJSON}

Save the media type.

TEST MEDIA TYPE

Media types list → Test

Use literal test values:
Subject: Workshop sink test
Message: service:checkout

Click Test → Open log
Expected response: logged

ASSIGN TO ADMIN

Users → Users → Admin
→ Media → Add

Type: Workshop log sink
Send to: local
When active: 1-7,00:00-24:00
Severities: all
Enabled: yes



3/4 · Create the action

The operation uses the Problem message template created in Step 2.

ACTION CONDITIONS

Alerts → Actions → Trigger actions

Event tag value:
service equals checkout
environment equals production
scope equals availability

Calculation: And

OPERATION

Operations → Add
Operation type: Send message
Send to users: Admin
Send to media type: Workshop log sink
Custom message: not selected

The operation uses the Problem message template from Step 2.



4/4 · Test production and staging

Use Reports → Action log as the final proof of routing and rendered tags.

PRODUCTION

zabbix_sender -z 127.0.0.1 -s tags-lab-01 -k workshop.event -o "FAIL payment-alert"

STAGING

zabbix_sender -z 127.0.0.1 -s tags-lab-02 -k workshop.event -o "FAIL payment-alert"

REPORTS → ACTION LOG — EXPECTED PRODUCTION RESULT

Message	Status	Info
Subject: Problem: Checkout component failure: FAIL payment-alert Message: Message type: Problem Subject: Checkout component failure: FAIL payment-alert Message: [{"tag":"instance","value":"payment-alert"}, {"tag":"scope","value":"availability"}, {"tag":"component","value":"api"}, {"tag":"environment","value":"production"}, {"tag":"region","value":"eu-central"}, {"tag":"service","value":"checkout"}]	Sent	

CLEAN UP

for h in tags-lab-01 tags-lab-02; do zabbix_sender -z 127.0.0.1 -s "\$h" -k workshop.event -o "RECOVER payment-alert"; done

Expected: production = Sent with Subject, Message and Tags JSON; staging = no action.



Task completed

Confirm the evidence before moving to the next topic.



PRACTICAL TASK 4 IS COMPLETE

PRODUCTION

Action log status: Sent

MESSAGE

Subject and Tags JSON are rendered.

STAGING

No action is created.

All practical tasks are complete. Finish with the application model and takeaways.



A simple rule for tag placement

Put a tag as high as you can, but only if it is true for everything below that level.

TEMPLATE

service:checkout
team:payments

HOST

environment:production
region:eu-central

ITEM

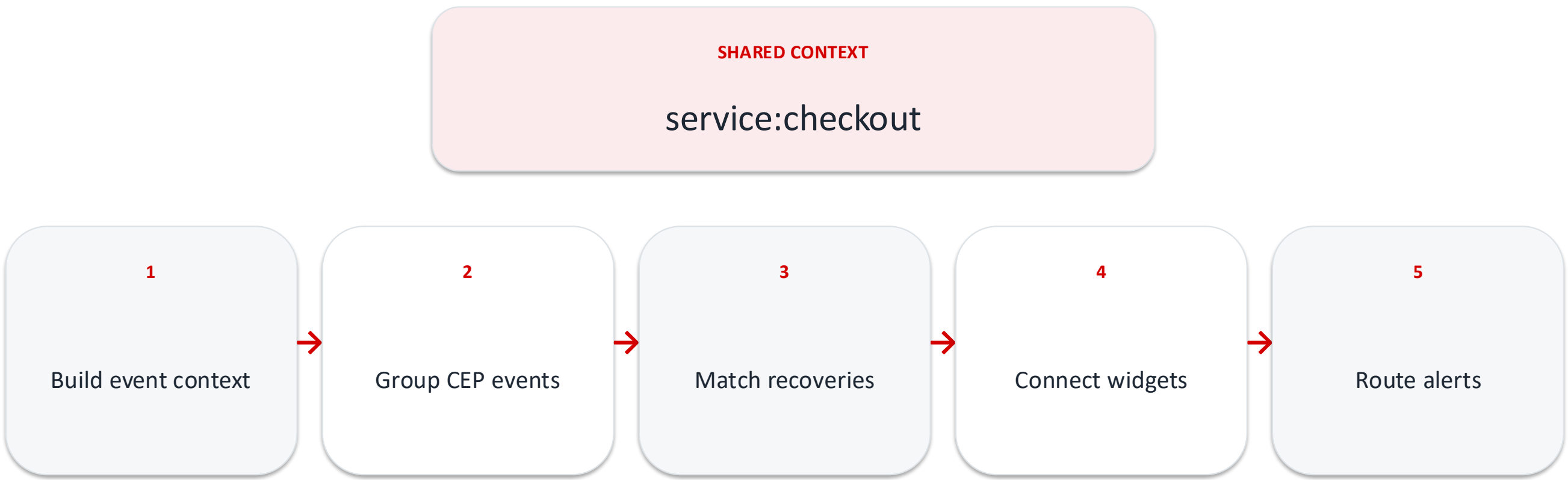
component:api
metric:latency

TRIGGER

scope:availability
instance:payment-01

One tag vocabulary, many uses

The same controlled metadata supports every practical task.



Design the vocabulary once. Reuse it consistently.

Thank you

Any questions?

WORKSHOP

Getting the Most Out of Tags

