

Complex event processing

Hands-on workshop

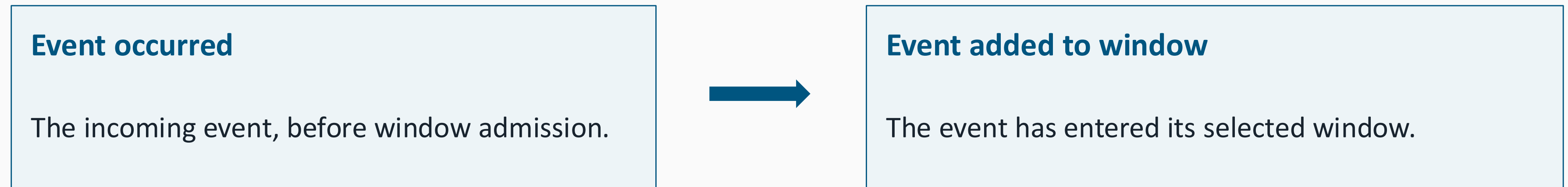
Five CEP modes. One prepared host.

Create rules and generate events with `zabbix_sender`.
Observe how CEP changes problems as events arrive.

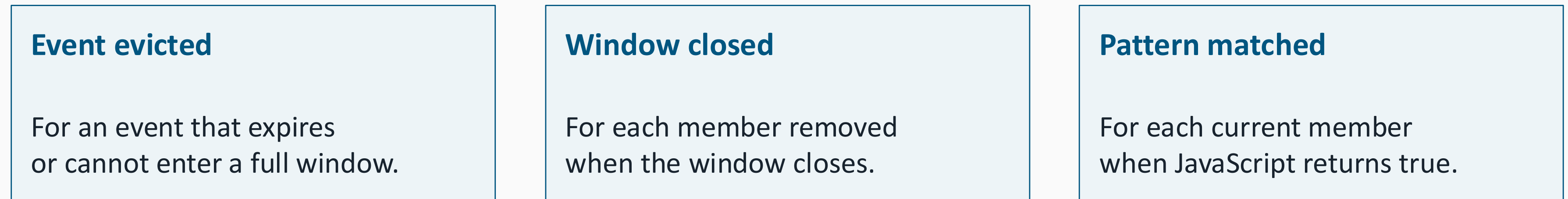
Host: workshop

When operations run

“Execute when” selects the moment and the events to process.



Later, the window can trigger different execution points



At full capacity, a new event can reach Event evicted without Event added.

The required order in the operation list

Keep operations grouped by “Execute when” in this order.

- 1 Event occurred
- 2 Event added to window
- 3 Event evicted
- 4 Window closed
- 5 **Pattern matched**

1–4: fixed order in the list

Each group runs at its own execution point.
An event can skip some stages.

5: a separate 1-second check

JavaScript checks the current window.
Operations run only if it returns true.

CEP windows in the database

Two read-only queries in the SQL client connected to the Zabbix database

1 Saved windows

```
SELECT w.cep_windowid, r.name AS rule,  
       w.hostid, COUNT(m.eventid) AS members  
FROM cep_window w  
LEFT JOIN cep_rule r  
  ON r.cep_ruleid = w.cep_ruleid  
LEFT JOIN cep_window_event m  
  ON m.cep_windowid = w.cep_windowid  
GROUP BY w.cep_windowid, r.name, w.hostid  
ORDER BY w.cep_windowid;
```

2 Members of one window

```
SELECT m.event_index, m.eventid,  
       e.name, e.clock, er.r_eventid  
FROM cep_window_event m  
LEFT JOIN events e  
  ON e.eventid = m.eventid  
LEFT JOIN event_recovery er  
  ON er.eventid = m.eventid  
WHERE m.cep_windowid = 12345  
ORDER BY m.event_index;
```

Replace 12345 with a window ID from query 1.

clock = Unix seconds. A recovery ID means the problem has closed.

Seven simple 0 / 1 items

Host: workshop · Zabbix trapper · Numeric (unsigned) · Warning triggers

Key	Problem name	Event tag
app	Application unavailable	service=customer-portal
disk	Disk latency high	service=storage
route	Checkout unavailable	service=checkout
uplink	Uplink down	service=branch-network
ping	Host unreachable	service=branch-network
web	HTTP check failed	service=branch-network
login	Login failed	service=authentication event=login_failed

Every trigger: last(/workshop/KEY)=1. Only login uses Multiple generation

LAB 1

Application ownership

None

Add a team tag when a new problem arrives.



Assign an application team

Conditions: Host = workshop AND tag service = customer-portal

Window: None

Execute when	Operation conditions	Do what
Event occurred	None	Add team=applications

Send zabbix_sender -z 127.0.0.1 -s workshop -k app -o 1

Reset zabbix_sender -z 127.0.0.1 -s workshop -k app -o 0

Result: Application unavailable has team=applications

LAB 2

Two rules, one event

Simple (multiple)

Assign a team and close an old storage problem.



Two rules for the same storage event

Both filters: Host = workshop AND tag service = storage

Both windows: Simple (multiple) Duration: 1m Unlimited Group by Host

Rule / sort order	Execute when	Operation conditions	Do what
WS2 Close old storage problem / 20	Event evicted	Problem is opened	1 Add closed_by=cep
WS2 Close old storage problem / 20	Event evicted	Problem is opened	2 Close the problem
WS2 Storage team / 21	Event added to window	None	Add team=storage

Both rules enabled. Stop after this rule unchecked in both.

One event enters both rule windows

Wait for both rule configurations to load before sending

Send `zabbix_sender -z 127.0.0.1 -s workshop -k disk -o 1`

Check	Expected result
After event processing	Disk latency high has team=storage
In the database, before expiry	The same event ID appears in both WS2 windows
At expiry in the closure rule	closed_by=cep is added, then the problem closes

After closure `zabbix_sender -z 127.0.0.1 -s workshop -k disk -o 0`

Two matching rules process one problem. Each rule owns a window.

LAB 3

One responsible team

Simple (once)

Use the first matching rule to select a team.



Choose one responsible team

Both rules: Host = workshop AND tag service = checkout

Simple (once) · 1m · Unlimited · Group by Host

Both rules match checkout · No operation conditions · Stop unchecked

Rule	Order	Execute when	Do what
WS3 Payments	30	Event added to window	Add team=payments
WS3 Fallback	31	Event added to window	Add team=noc

Send zabbix_sender -z 127.0.0.1 -s workshop -k route -o 1

Reset zabbix_sender -z 127.0.0.1 -s workshop -k route -o 0

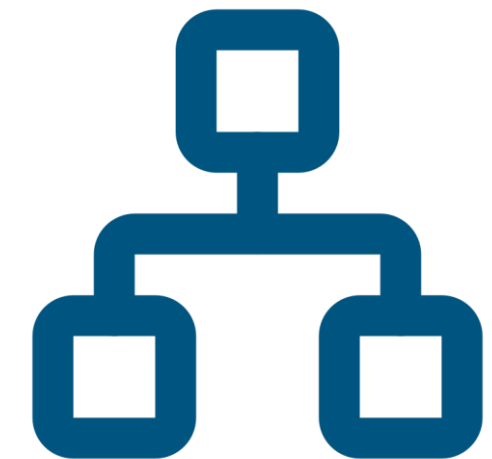
Both filters match. The event gets team=payments only

LAB 4

One network incident

Cause and symptoms grouping

Link two symptoms and count them on the cause.



A network outage with a symptom count

Conditions: Host = workshop AND tag service = branch-network

Duration: 2m Capacity: Unlimited Group by Host

Event count tag: Yes Tag name: symptom_count

Execute when	Operation conditions	Do what
Event added to window	None	Add incident=network

Send uplink first. When it appears, send ping and web in the same period.

```
zabbix_sender -z 127.0.0.1 -s workshop -k uplink -o 1
zabbix_sender -z 127.0.0.1 -s workshop -k ping -o 1
zabbix_sender -z 127.0.0.1 -s workshop -k web -o 1
```

The cause counts its symptoms

The first event becomes the cause. Each later event adds a symptom

New event	Role	Tag on Uplink down
Uplink down	Cause	No count tag yet
Host unreachable	First symptom	symptom_count=1
HTTP check failed	Second symptom	symptom_count=2

All three events have incident=network.
Only the cause receives the symptom_count tag.

Two symptoms give a count of 2. The cause itself is excluded.

LAB 5

Repeated login failures

Pattern match

Create one incident and suppress repeated failures.



One item, multiple problem events

Three login failures create one new incident

Item and trigger	Setting
Host and item key	workshop / login
Trigger expression	last(/workshop/login)=1
PROBLEM event generation	Multiple
OK event closes	All problems

Conditions: Host = workshop AND service = authentication
AND event = login_failed

Duration: 1m Capacity: Unlimited Group by Host

Three original events in the window

Paste pattern.js into the rule

```
var events = cep_get_events();
var failures = 0;

for (var i = 0; i < events.length; i++) {
    var event = events[i];
    if (!event.is_copied)
        failures++;
}

return failures >= 3 ? 'true' : 'false';
```

Count originals

Skip cloned events.
Return true at three
or more originals.

Keep the lab simple

Recovered originals
also count until expiry.
Send 0 only for reset.

Clone first, then suppress originals

Use this exact list order. Combine operation conditions with And

Order	Execute when	Operation conditions	Do what
1	Event occurred	Event is cloned	Set name: Repeated login failures
2	Pattern matched	Event is not cloned Problem is opened Problem is not suppressed	Clone first
3	Pattern matched	Event is not cloned Problem is opened Problem is not suppressed	Suppress indefinitely

Keep the window open. The clone stays unsuppressed

Three attempts, then one more

Use the same item. Do not send 0 between attempts

```
zabbix_sender -z 127.0.0.1 -s workshop -k login -o 1  
zabbix_sender -z 127.0.0.1 -s workshop -k login -o 1  
zabbix_sender -z 127.0.0.1 -s workshop -k login -o 1
```

After the third attempt and CEP processing:

One visible clone: Repeated login failures

All three Login failed originals are suppressed

Fourth attempt `zabbix_sender -z 127.0.0.1 -s workshop -k login -o 1`

The fourth original is suppressed. The same clone remains

Window expiry and replay

Further suppression requires at least three originals in the window

While the count is 3 or more

A matching evaluation suppresses new original problems.
The existing clone stays visible.

When the count drops below 3

New originals stay unsuppressed until the count reaches 3 again.
Existing suppression continues.

Reset `zabbix_sender -z 127.0.0.1 -s workshop -k login -o 0`

All problems recovery closes the originals and the clone.
Before replay, wait for an empty window or recreate this lab rule.