

# Actions and Automation Overview



Wiz integrates with a [wide variety of third-party tools](#) so that you can automatically run actions on newly detected Issues and cloud events, export information from Wiz, and enrich the Security Graph with the results generated by other tools. Alternatively, you can create your own [custom integration](#).


## Integrations, Actions, and Automation Rules

There are three distinct functional components involved in an automation flow: an Integration, an Action and (optionally) an Automation Rule:

Component	Description	Example(s)
Integration	<p>There are four types of integrations:</p> <ul style="list-style-type: none"> <li>• Push</li> <li>• Pull</li> <li>• Export</li> <li>• Enrich</li> </ul> <p>Actions and automation refer to Push integrations only. This component is the authentication info required for Wiz to communicate with the third-party tool. <a href="#">Learn about other integrations</a>.</p>	<ul style="list-style-type: none"> <li>• Username and password</li> <li>• URL and token</li> </ul>
Action	<p>The information sent from Wiz to the third-party tool using the Integration. There are several ways to <a href="#">run an Action</a>.</p>	<ul style="list-style-type: none"> <li>• A JSON describing a new Issue</li> <li>• Custom headers and request body for a webhook</li> </ul>
Automation Rule	<p>A set of conditions (WHEN, IF, THEN) linking a change in the state of an Issue or Cloud Event with an Integration and an Action.</p>	<p>WHEN an Issue is created, IF it is Critical severity, THEN send a Slack message to the #cloud-sec channel.</p>

⚠️ Automation Rules are subject to limits. [Learn more](#).

The authentication info saved as an integration can be re-used by any number of Actions and/or Automation Rules.

 Webhook Integration

### Manually run an Action on an Issue

Public bucket with sensitive data

Add Note Run an action Create a Ticket Share Feedback

### Manually run an Action on many Issues

Issues GROUP BY Type Resource Subscription None

public bucket with sensitive di Risk Category Subscription Severity Project

Issue	Resource	Risks
Public bucket with sensitive data	liron-public-b- S3 Bucket	
Public bucket with sensitive data	moran-bucket- S3 Bucket	
Public		

3 issues selected Clear Run action(s) Update...

### Run an Action via Automation Rule(s)

Webhook Issues	Issues	Dec 11, 2022 at 2:02 PM
Webhook - Webhook Issues	Created, Reopened, Resolved, Updated	
Webhook Cloud Events	Cloud Events	Nov 30, 2022 at 7:52 PM
Webhook - Cloud Events	Created	

*One webhook Integration, many uses.*

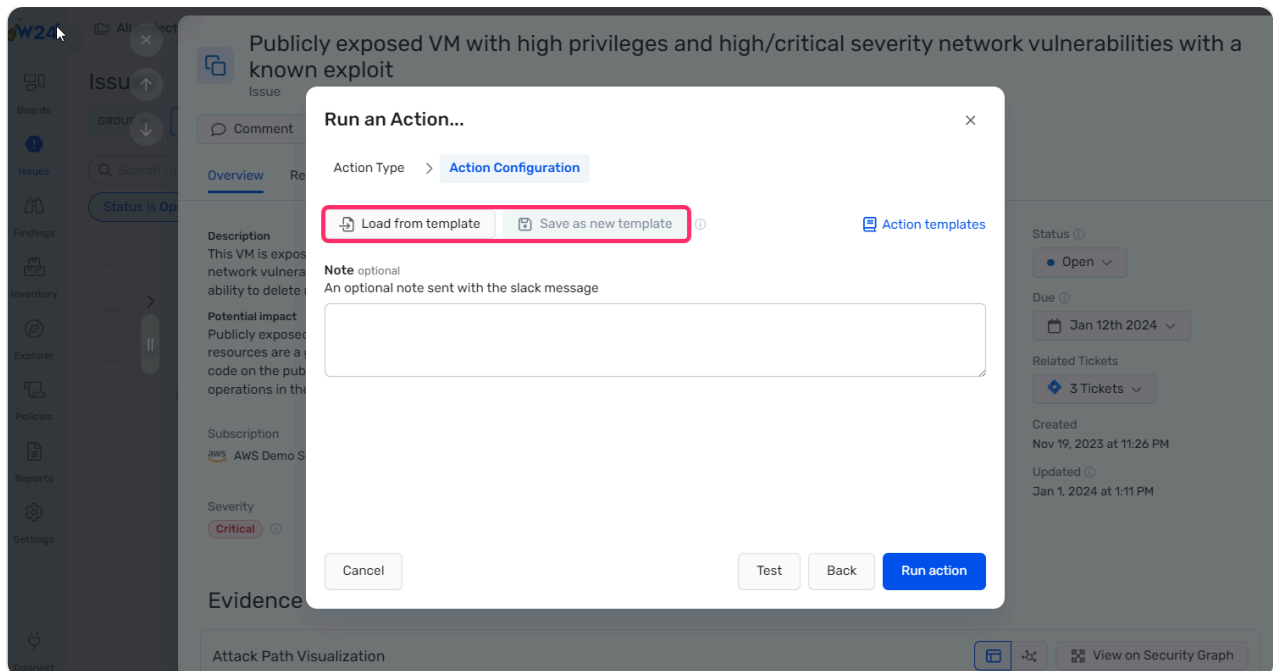
To better understand why this division is so powerful, see the detailed [sample use cases](#).

## Actions vs. Action templates

An integration on its own is not especially useful because it contains only the info required to communicate with a third-party tool. A webhook integration, for instance, is only a URL, additional headers, and authentication details like username/password. A webhook Action, on the other hand, is additional headers and a request body.

An Action template is a set of saved Action parameters that can be re-used by many Automation Rules. You can use, create and delete Action templates in two places:

- On the Issues page, when manually running an Action on an Issue (or running an Action on multiple Issues):



- On the Policies > Automation Rules page, when adding or editing an Automation Rule.

Every Automation Rule has the same options for its basic info and Rule conditions. The Action Configurations are specific to the selected Integration / Action type(s).



- Action templates cannot be modified; only duplicated and deleted.
- Deleting an Action template has no effect on any Automation Rules that originally used the deleted Action template.

## Running Actions

Actions can be run either manually from the Wiz portal or automatically using Automation Rules.

- On the Issues page, you can [run an Action on a single Issue](#), on [multiple Issues in bulk](#), or on [a Control](#).
- On the [Policies > Automation Rules](#) page, you can define Automation Rules that run an Action when matching Issues are generated by a Control or cloud events are detected by a Threat Detection Rule. This is how [Auto-Remediation](#) is achieved without a "human in the loop."

## Testing integrations, Actions, & Automation Rules

There are several ways to test integrations, Actions, and Automation Rules:

- Integration—[Validate auth info only](#).
- Action—Perform an end-to-end test [using mock data](#), or manually [run an Action on an Issue](#).
- Automation Rule—Perform an end-to-end test [using mock data](#).

### ▼ **Validate auth info only**

To test only the connectivity of an integration:

1. On the ⚙ Settings > Integrations page, click More options > Test run. Wiz validates the authentication info.
2. Inspect the result and, if it failed, check the authentication info associated with the integration.

### ▼ **Using mock data**

⚠ Testing Actions triggered by Cloud Events is not yet supported.

Wiz stores mock data to make it easier to test Actions and Automations Rules:

Click to expand ☐ Issue mock data    Control mock data

To test an Action using mock data:

1. On the Issues page, start [running an Action on an Issue](#).
2. At the very end, click Test (instead of clicking Run Action).
3. If the test does not work as expected, inspect the error message, validate it against the mock data, then check the integration's auth info and/or the Action configuration.

To test an Automation Rule using mock data:

1. Start [adding an Automation Rule](#).
2. When defining the Rule Conditions, after selecting an existing integration and configuring the Action Type and Action Configuration, click Test. The Automation Rule is triggered immediately using mock data.
3. If the test works as expected, click Add Rule. Otherwise, inspect the error message, validate it against the mock data, then check the integration's auth info and/or the Automation Rule configuration.

## **Automation Rule triggers**

When defining an Automation Rule, you must specify WHEN the Automation Rule should be triggered.

Automation Rules can be triggered by different combinations of sources and statuses. Some combinations are not supported. Moreover, some combinations are not supported for all integrations.

--	--	--

Issues	Created	A Control detected an active risk on a resource for the first time
	Reopened	A Control detected an active risk on a resource where the same Control previously detected a risk
	Resolved	A Control no longer detects a risk on a resource
	Updated	<ul style="list-style-type: none"> <li>A user manually changed the status of an Issue</li> <li>Wiz re-defined the Control that generated the Issue</li> </ul>
	Due	The due date assigned to an Issue has expired or is going to be expired in X days
Cloud events	Matched	A cloud event is matched by a rule
Controls	Created, Reopened	—
	Resolved	The number of Open and In Progress Issues associated with a Control decreases to zero
	Updated	The number of Open and/or In Progress Issues associated with a Control increases or decreases to a non-zero value
Audit logs	Created	A new audit log is created
System Health Issues	Created	—
	Updated	Wiz redefines the name or severity of the System Health Issues, or, for a System Health Issue that aggregates several errors, a change in the Issue scope
	Resolved	A System Health Issue is manually resolved or resolved by Wiz

The tables of [supported third-party tools](#) indicate which Integrations can be triggered by Issues, Cloud Events, and/or Controls.

## Project- and role-based scoping for Integrations and Automation Rules

When adding an integration, you can choose a Project- and role-based scope to limit where and by whom the integration can be used:

- All resources—The integration can be used in every project scope by any user with write permissions for Actions and Automation Rules, e.g. Project Admins, Global Contributors, and Global Admins.
- If you select All Resources, you can further restrict the Integration to global roles only to prevent users with Project-scoped roles from seeing or using the integration. Furthermore, only Automation Rules with the All Resources scope can use such Integrations.
- Selected Project—The integration can be used only in the selected Project by users with write permissions for Settings > Integrations pages.

For Automation Rules that are triggered by changes to Controls, the scope of the Action is based on the properties chosen for the Automation Rule in the following order:

Source	Status	Description
--------	--------	-------------

2. The Project filter of the Automation Rule. If none is set, then...
3. The Project scope of the Control. If none is set, then...
4. The All Resources scope.

For more information about user roles in Wiz, see the guide on [Role-Based Access Control](#).

**i** System Health Issues do not support project and role-based scoping.

## IP access control

Many organizations apply IP access control on the environment or application level to provide an additional layer of security when they integrate with tools like Wiz. If your organization applies IP access control, you must allow the relevant Wiz [scanner IP addresses](#).

All API and webhook calls originate from the scanner IP addresses for your tenant.

## Wiz template variables

Integrations that [support template variables](#) allow you to take variables from the Wiz platform to enrich the information sent as an Action.

### ▼ Template variables for Issues

These variables are divided into three categories:

- [Issues fields](#)—The Issue itself
- [Platform fields](#)—Info about the Control that generated the Issue
- [Object fields](#)—Info about the resource (VM, container, etc.) on which Wiz detected the Issue

#### Issue fields

- issue.id
- issue.status
- issue.createdAt
- issue.updatedAt
- issue.resolvedAt
- issue.description
- issue.severity
- issue.resolutionReason
- issue.type
- issue.dueAt

- `{{#issue.notes}}{{user.email}}-{{text}}, {{/issue.notes}}` —Notes refers to the comments tab in the Issues drawer, including the ignore information, namely: Ignore Reason, Ignore Comment and Ignore Until date.
- `issue.evidence`—The evidence json includes following graph objects: Finding, Configuration Finding, Hosted Technology, Malware Instance, Secret Instance, Access Role Permission, User Account, Service Account, Endpoint, Security Event Finding, Lateral Movement Finding, Namespace, Kubernetes Cluster, Authentication Configuration, and File Descriptor Finding.
- `{{#issue.serviceTickets}}{{name}}, {{/issue.serviceTickets}}` —List of service tickets on an Issue.
- `{{#issue.projects}}{{name}}, {{/issue.projects}}` —List of Projects in Wiz that this Issue is part of. You can combine this field with more attributes such as:
  - `{{#issue.projects}}{{name}}:{{description}}, {{/issue.projects}}` —Project name and description.
  - `{{#issue.projects}}{{name}}-owners:{{#projectOwners}}{{email}}, {{/projectOwners}}{{/issue.projects}}` —Project name and owners (email addresses).
  - `{{#issue.projects}}{{name}}-business impact:{{#riskProfile}}{{businessImpact}}, {{/riskProfile}}{{/issue.projects}}` —Project name and business impact.
- `{{issue.entitySnapshot.createdAt}}` —Creation time of the primary resource.
- `{{#changedFields}} {{name}} field was changed from {{previousValue}} to {{newValue}}, {{/changedFields}}` —List of updated fields that triggered the Action via Automation Rule with "Updated" filter.
- `{{changedBy}}`
- `triggeredBy` —The user or the Automation Rule that ran an Action.

**i** All template variables should be surrounded by double curly braces, e.g. `{{templateVariable}}`, in an integration. Only the `issue.notes`, `issue.serviceTickets`, `changedFields`, and `issue.projects` template variables require pound sign ( # ) and forward slash ( / ) as well.

## Platform fields

- `issue.control.descriptionPlainText`
- `issue.control.id`
- `issue.control.name`
- `issue.control.resolutionRecommendation`
- `issue.control.resolutionRecommendationPlainText`
- `issue.control.sourceCloudConfigurationRule.id`
- `issue.control.sourceCloudConfigurationRule.name`

## Object fields

- `issue.entitySnapshot.providerId`

- `issue.entitySnapshot.type`
- `issue.entitySnapshot.nativeType`
- `issue.entitySnapshot.name`
- `issue.entitySnapshot.subscriptionExternalId`
- `issue.entitySnapshot.subscriptionName`
- `issue.entitySnapshot.subscriptionTags`
- `issue.entitySnapshot.resourceGroupExternalId`
- `issue.entitySnapshot.region`
- `issue.entitySnapshot.cloudPlatform`
- `issue.entitySnapshot.status`
- `issue.entitySnapshot.cloudProviderURL`
- `issue.entitySnapshot.tags`
- `issue.entitySnapshot.cloudProviderOriginalJson`
- `issue.entitySnapshot.externalId`
- `issue.entitySnapshot.kubernetesClusterId`
- `issue.entitySnapshot.kubernetesClusterName`
- `issue.entitySnapshot.kubernetesNamespaceName`
- `issue.entitySnapshot.containerServiceId`
- `issue.entitySnapshot.containerServiceName`



- Template variables for tags send a list of all tags stored on the resource or the subscription. For a specific tag you can explicitly add its key, e.g., if you want the value of the tag "Owner" on the resource use the template variable `issue.entitySnapshot.tags.Owner` (or `issue.entitySnapshot.subscriptionTags.Owner` for tags on the resource subscription)
- Template variables `issue.evidence` , `issue.entitySnapshot.tags` , `issue.entitySnapshot.subscriptionTags` , and `issue.entitySnapshot.cloudProviderOriginalJson` have a JSON structure, thus when added to a request body they should not be wrapped by quotation marks.
- Template variable dates are sent in ISO 8601 format (YYYY-MM-DDThh:mm:ss.sTZD)
- If you want to add a link back to the portal to view the Issue, use the following string `"https://{wizDomain}}/issues#~(issue~'{{{issue.id}}})"`

### ▼ Template variables for Cloud Events

These variables are divided into three categories: [Cloud event fields](#), [Resource fields](#), and [Matched rules](#) (the resource which Wiz detected the issue on).

#### Cloud event fields



- `cloudEvent.id`
- `cloudEvent.name`
- `cloudEvent.cloudPlatform`
- `cloudEvent.timestamp`
- `cloudEvent.source`
- `cloudEvent.category`
- `cloudEvent.actor.id`
- `cloudEvent.actor.externalId`
- `cloudEvent.actor.name`
- `cloudEvent.actor.type`
- `cloudEvent.actorIP`
- `cloudEvent.actor.actingAs.id`
- `cloudEvent.actor.actingAs.name`
- `cloudEvent.actor.actingAs.providerUniqueId`
- `cloudEvent.actor.actingAs.type`
- `cloudEvent.path`—This is a JSON array and shouldn't be surrounded by quotes
- `cloudEvent.rawAuditLogRecord`—The raw body of the original cloud event

## Resource fields

- `cloudEvent.subjectResource.id`
- `cloudEvent.subjectResource.name`
- `cloudEvent.subjectResource.nativeType`
- `cloudEvent.subjectResource.providerUniqueId`
- `cloudEvent.subjectResource.externalId`
- `cloudEvent.subjectResource.region`
- `cloudEvent.subjectResource.cloudAccount.name`
- `cloudEvent.subjectResource.cloudAccount.externalId`
- `cloudEvent.subjectResource.cloudAccount.cloudProvider`
- `cloudEvent.subjectResource.kubernetesCluster.name`
- `cloudEvent.subjectResource.kubernetesCluster.id`
- `cloudEvent.subjectResource.kubernetesFlavor`
- `cloudEvent.subjectResource.kubernetesNamespace.name`
- `cloudEvent.subjectResource.kubernetesNamespace.id`

## Matched rules

A list of all Threat Detection Rules that apply to this cloud event:

```
{{#cloudEvent.matchedRules}} ruleId: {{rule.id}}; ruleSeverity: {{rule.severity}};
ruleName: {{rule.name}}; {{/cloudEvent.matchedRules}}
```

**i** All template variables should be surrounded by double curly braces, e.g. `{{templateVariable}}`, in an integration. Only

the `cloudEvent.matchedRules` template variables requires a pound sign ( # ) and forward slash ( / ) as well.

#### ▼ Template variables for Controls

- `control.id`
- `control.name`
- `control.description`
- `control.descriptionPlainText`
- `control.type`
- `control.severity`
- `control.enabled`
- `control.enabledForLBI`
- `control.enabledForMBI`
- `control.enabledForHBI`
- `control.resolutionRecommendation`
- `control.resolutionRecommendationPlainText`
- `control.createdBy`
- `control.createdAt`
- `control.scopeProject`
- `control.sourceCloudConfigurationRule`
- `control.lastRunAt`
- `control.lastSuccessfulRunAt`
- `{{#control.serviceTickets}}{{name}}, {/control.serviceTickets}}`
- `scopedControlRun.createdIssuesCount`
- `scopedControlRun.reopenedIssuesCount`
- `scopedControlRun.rejectedIssuesCount`
- `scopedControlRun.resolvedIssuesCount`
- `projectFilterUrl` - the project scope of which the Action was triggered by
- `triggeredBy`—The user who manually ran an Action or the Automation Rule that automatically ran an Action

**⚠** To send the URL to view the Issues related to the Control scope run, use the following string:

```
https://{{wizDomain}}/issues#~(filters~(control~(equals~('{{control.id}})){{projectFilterUrl}})~groupBy~'none)
```

#### ▼ Template variables for audit logs

- `auditLogEntry.id`

- `auditLogEntry.status`
- `auditLogEntry.action`
- `auditLogEntry.actionParameters`
- `auditLogEntry.requestId`
- `auditLogEntry.user`
- `auditLogEntry.serviceAccount`
- `auditLogEntry.timestamp`
- `auditLogEntry.sourceIP`
- `auditLogEntry.userAgent`
- `auditLogEntry.status`
- `auditLogEntry.duration`



- Action templates for audit log are only supported for AWS SNS, Azure Service Bus, and GCP Pub/Sub.
- Action templates for audit logs require a role that has admin permissions on the audit log.
- All template variables should be surrounded by double curly braces, e.g. `{{templateVariable}}`, in an integration.
- Template variables `auditLogEntry.actionParameters`, `auditLogEntry.user`, and `auditLogEntry.serviceAccount` have a JSON structure, thus when added to a request body they should not be wrapped by quotation marks.
- Template variable dates are sent in ISO 8601 format (YYYY-MM-DDThh:mm:ss.STZD)

#### ▼ **Template variables for System Health Issues**

- `shi.name`
- `shi.severity`
- `shi.deployment.id`
- `shi.deployment.name`
- `shi.source.externalId`
- `shi.impact`
- `shi.remediation`
- `shi.affectedModules`



All template variables should be surrounded by double curly braces, e.g. `{{templateVariable}}`, in an integration.

## Sample use cases

The separation of automation flows into three distinct functional components—Integrations, Actions, and Automation Rules—enables a wide variety of use cases. Here are a few to get you started:

#### ▼ **One Jira, many projects**

Many companies use Jira for ticketing, to plan development, track patching, etc. The larger your organization is, the more projects in Jira you probably have. Imagine you have different projects for dev teams A and B and security teams Y and Z.

One way to automate your responses to Issues detected by Wiz would be:

1. Add a single Integration for Jira
2. Save a separate Action template for each dev and security team
3. Add multiple Automation Rules that meet each team's operational requirements and SLAs. For instance:
  - If dev team A is responsible for production infrastructure and dev team B is responsible for test infrastructure, add one Automation Rule that creates a Jira ticket on the Jira board owned by team A when critical and high severity Issues are detected in the production environment, and add a second Automation Rule that creates a ticket on the Jira board of team B when Issues are detected in the test environment.
  - If dev team B is responsible for several different applications, add different Automation Rules that create Jira tickets based on the sensitivity and business importance of the apps. Issues detected on cloud resources that support a financial processing app should probably be remediated before Issues in a QA sandbox, so you can create an Action template that creates Jira tickets with severity HIGHEST for the financial processing app and another Action template that creates Jira tickets with severity MEDIUM when an Issue is detected in the QA sandbox.
  - If security team Y is responsible for patching vulnerabilities but security team Z is responsible for IAM, add Automation Rules that create Jira tickets in the relevant Jira project based on Issue categories.

These are just hypothetical examples. You know your business and your cloud estate better than we do. The point is that Integrations, Actions, and Automation Rules together form a tremendously flexible and powerful system for automating response and routing information.

#### ▼ **Periodic key rotation**

All other things being equal, the more sensitive a tool is, the more often you should rotate any access keys granted to third parties, including Wiz. By saving each tool's authentication information only once as an Integration, and then defining numerous Action templates and Automation Rules based on that Integration, you can dramatically reduce the effort involved in periodically rotating those access keys.

To efficiently manage key rotation for third-party tools integrated with Wiz:

1. Consolidate Actions that were migrated to Integrations such that each set of authentication info (username/password, token, etc.) is stored in only one Integration in Wiz.
2. In your third-party tool, generate new authentication info, then copy it to a local file.

❗ Generating new authentication info probably breaks the Integration in Wiz. Be sure to update the Integration right away.

3. In Wiz, on the Settings > Integrations and Automation > Integrations page, click the corresponding Integration.
4. Paste the new authentication info.
5. Click Save.
6. (Recommended) Test the updated Integration by manually [running an Action](#).

### ▼ Integrating with on-prem tools

Due to regulations and existing infrastructure, many companies use self-hosted tools like Jira and Splunk. For security reasons, these tools might not be accessible from the public internet. Nonetheless, a Wiz Broker can be used to integrate such tools with Wiz:

- Add a single Integration for the on-prem tool. If you enable This server is only accessible on-premises, you will be prompted to deploy a Wiz Broker after you add the Integration. [Learn about the Wiz Broker](#).
- Add any number of Action templates and Automation Rules based on the single Integration and Wiz Broker.

### ▼ Many new Issues, one new ticket

In large and dynamic environments, defining an Automation Rule that generates a new ticket in Jira, ServiceNow, etc. per Issue can overwhelm security teams. It can also create redundancy if a similar risk affects multiple hosts. Take, for example, a mistake in an IaC definition that causes dozens of buckets to be accidentally publicly exposed.

You can alleviate alert fatigue and reduce redundancy by scoping Automation Rules to Controls instead of Issues. An Automation Rule scoped to Controls is triggered when the number of Open and In progress Issues associated with a Control changes, and only once per Control assessment cycle. For instance, if the Control for [CVE-2021-44228 \(Log4Shell\) detected on a publicly exposed VM instance/serverless](#) (wc-id-609) had zero Issues associated with it on Monday but five Open Issues associated with it on Tuesday, an Automation Rule scoped to Controls would be triggered only once. If the same Control had only one In Progress Issue on Wednesday, the Automation Rule would again be triggered only once.

To add an Automation Rule scoped to Controls:

1. Add an integration for your third-party ticketing service, e.g. Jira or ServiceNow.

2. Add an Automation Rule with the following rule conditions:
  - WHEN—Controls are Updated
  - IF—Select relevant criteria, e.g. sensitive Projects or Controls that frequently generate multiple Issues
  - THEN—Select the integration with your third-party ticketing and the create ticket Action
3. Select or provide the Integration-specific Action configurations, e.g. Project Key, Table Name, etc.
4. Click Add Rule.

## Supported third-party tools

The following tables list the third-party tools with which Wiz can integrate for Actions and Automation Rules (Push Integrations); whether they can be triggered by Issues, Cloud Events, Controls, audit logs, and/or System Health Issues (SHI); and whether they support [template variables](#). For other integration types, [see all supported third-party tools](#).

<a href="#">Amazon SQS</a>	☐	☐	☐	☐	☐	☐
<a href="#">AWS EventBridge</a>	☐	☐	☐	☐	☐	☐
<a href="#">AWS SNS</a>	☐	☐	☐	☐	☐	☐
<a href="#">Azure DevOps Ticketing</a>	☐	☐	☐	☐	☐	☐
<a href="#">Azure Logic Apps</a>	☐	☐	☐	☐	☐	☐
<a href="#">Azure Sentinel</a>	☐	☐	☐	☐	☐	☐
<a href="#">Azure Service Bus</a>	☐	☐	☐	☐	☐	☐
<a href="#">Botprise</a>	☐	☐	☐	☐	☐	☐
<a href="#">Cisco Webex</a>	☐	☐	☐	☐	☐	☐
<a href="#">ClickUp</a>	☐	☐	☐	☐	☐	☐
<a href="#">Cortex XSOAR</a>	☐	☐	☐	☐	☐	☐
<a href="#">CyberArk</a>	☐	☐	☐	☐	☐	☐
<a href="#">Cyware</a>	☐	☐	☐	☐	☐	☐
<a href="#">Email</a>	☐	☐	☐	☐	☐	☐
<a href="#">Fortinet</a>	☐	☐	☐	☐	☐	☐
<a href="#">Freshservice</a>	☐	☐	☐	☐	☐	☐
<a href="#">Google Chat</a>	☐	☐	☐	☐	☐	☐

<a href="#">Google Cloud Chronicle</a>	☐	☐	☐	☐	☐	☐
<a href="#">Google Cloud Pub/Sub</a>	☐	☐	☐	☐	☐	☐
<a href="#">Hunters</a>	☐	☐	☐	☐	☐	☐
<a href="#">Illumio</a>	☐	☐	☐	☐	☐	☐
<a href="#">Jira</a>	☐	☐	☐	☐	☐	☐
<a href="#">Linear</a>	☐	☐	☐	☐	☐	☐
<a href="#">Microsoft Teams</a>	☐	☐	☐	☐	☐	☐
<a href="#">Netskope</a>	☐	☐	☐	☐	☐	☐
<a href="#">Opsgenie</a>	☐	☐	☐	☐	☐	☐
<a href="#">Opus</a>	☐	☐	☐	☐	☐	☐
<a href="#">PagerDuty</a>	☐	☐	☐	☐	☐	☐
<a href="#">ServiceNow Ticketing</a>	☐	☐	☐	☐	☐	☐
	☐	☐	☐	☐	☐	<b>Template variables</b>
	<b>Issues</b>	<b>Cloud Events</b>	<b>Controls</b>	<b>Audit logs</b>	<b>SHI</b>	
<a href="#">Tines</a>	☐	☐	☐	☐	☐	☐
<a href="#">Torq</a>	☐	☐	☐	☐	☐	☐
<a href="#">Webhook</a>	☐	☐	☐	☐	☐	☐

## Troubleshooting

On the [Settings > System Activity Log](#) page, you can view Integration Actions that were triggered and their statuses.

- Click View Errors to investigate failed integration Actions runs.
- Click View Full Context to inspect the Action parameters.

### ▼ `{{evidence}}` template variable is not being sent

The `{{evidence}}` template variable is only sent when the Automation Rule filters alert on `Created` or `Reopened` Issues.

### ▼ `{{evidence}}` or `{{entitySnapshot.cloudProviderOriginalJson}}` template variables generate an invalid JSON

The evidence and entitySnapshot.cloudProviderOriginalJson template variables are JSON objects, not a string representation of the JSON object, so they should be used without wrapping them with double quotes, i.e., `{{evidence}}` and not `"{{evidence}}"`.

Issues	Cloud Events	Controls	Audit logs	SHI	Template variables
--------	--------------	----------	------------	-----	--------------------

These errors can occur when trying to trigger an Action on a resource that was already deleted from the Security Graph. First verify that the Issue and the resources related to it still exist on the Security Graph. If they do, [let us know](#).

#### ▼ In integrations for Cloud Events, the link back to the Wiz portal leads to an empty event

Threat Detection Rules are processed against the raw JSON of the event, which is normalized by Wiz. This could create a lag of up to 15 minutes between the initial assessment of the Threat Detection Rule and the time it takes for the event to actually appear in the Wiz portal.

#### ▼ How to set a default value for a template variable in case it does not return a value

Template variables leverage a limited implementation of the mustache library, so you can use the implicit operator `{{.}}` and inverted sections to return a default value if a template variable has no value to return.

Format:

```
"tags":{{#TEMPLATE-VARIABLE}}{{.}}{{/TEMPLATE-VARIABLE}}{{^TEMPLATE-VARIABLE}}DEFAULT-VALUE{{/TEMPLATE-VARIABLE}}
```

Example:

```
"tags":{{#issue.entitySnapshot.tags}}{{.}}{{/issue.entitySnapshot.tags}}{{^issue.entitySnapshot.tags}}{{/issue.entitySnapshot.tags}}
```

In the above example, the tags template variable did not return a value because the resource didn't have any tags. The default value is returned, which is an empty JSON object `{}`. This means that you can guarantee that a valid JSON is returned even when no value is returned from a template variable.

## FAQ

Questions? See the [Response and Automation FAQ](#).



---

[← Supported Techs for Application Fingerprinting Pa-Z](#)

[Update Integrations Running AWS Lambda due to CVE-2024-30251 →](#)

Did this page help you? ☒ **Yes** ☐ **No**