AWS Fault Injection Simulator

User Guide

AWS Fault Injection Simulator: User Guide

Table of Contents

What is AWS Fault Injection Simulator?

AWS Fault Injection Simulator (AWS FIS) is a managed service that enables you to perform fault injection experiments on your AWS workloads. Fault injection is based on the principles of chaos engineering. These experiments stress an application by creating disruptive events so that you can observe how your application responds. You can then use this information to improve the performance and resiliency of your applications so that they behave as expected.

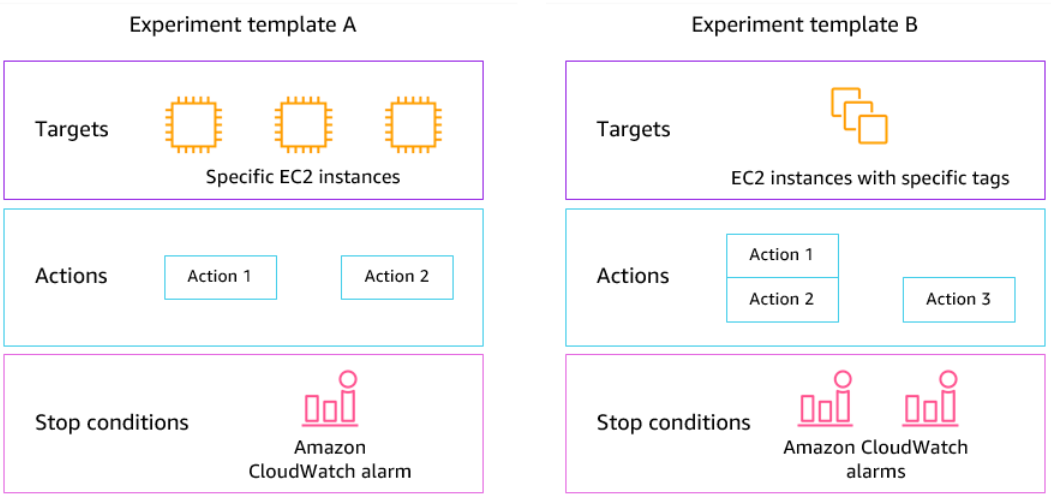
To use AWS FIS, you set up and run experiments that help you create the real-world conditions needed to uncover application issues that can be difficult to find otherwise. AWS FIS provides templates that generate disruptions, and the controls and guardrails that you need to run experiments in production, such as automatically rolling back or stopping the experiment if specific conditions are met.

> **Important**
> AWS FIS carries out real actions on real AWS resources in your system. Therefore, before you use AWS FIS to run experiments in production, we strongly recommend that you complete a planning phase and run the experiments in a pre-production environment.

For more information about planning your experiment, see Test Reliability and Plan for experiments (p. 4). For more information about AWS FIS, see AWS Fault Injection Simulator.

AWS FIS concepts

To use AWS FIS, you run *experiments* on your AWS resources to test your theory of how an application or system will perform under fault conditions. To run experiments, you first create an *experiment template*. An experiment template is the blueprint of your experiment. It contains the *actions*, *targets*, and *stop conditions* for the experiment. After you create an experiment template, you can use it to run an experiment. While your experiment is running, you can track its progress and view its status. An experiment is complete when all of the actions in the experiment have run.



Actions

An *action* is an activity that AWS FIS performs on an AWS resource during an experiment. AWS FIS provides a set of preconfigured actions based on the type of AWS resource. Each action runs for a specified duration during an experiment, or until you stop the experiment. Actions can run sequentially or simultaneously (in parallel).

Targets

A *target* is one or more AWS resources on which AWS FIS performs an action during an experiment. You can choose specific resources, or you can select a group of resources based on specific criteria, such as tags or state.

Stop conditions

AWS FIS provides the controls and guardrails that you need to run experiments safely on your AWS workloads. A *stop condition* is a mechanism to stop an experiment if it reaches a threshold that you define as an Amazon CloudWatch alarm. If a stop condition is triggered while the experiment is running, AWS FIS stops the experiment.

AWS FIS actions and supported AWS services

AWS FIS provides preconfigured actions for specific types of targets across AWS services. For more information, see Actions for AWS FIS (p. 12).

AWS FIS supports actions for target resources for the following AWS services:

- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Relational Database Service (Amazon RDS)

AWS FIS access

You can work with AWS FIS in any of the following ways:

- **AWS Management Console** — Provides a web interface that you can use to access AWS FIS. For more information, see Working with the AWS Management Console.
- **AWS Command Line Interface (AWS CLI)** — Provides commands for a broad set of AWS services, including AWS FIS, and is supported on Windows, macOS, and Linux. For more information, see AWS Command Line Interface.
- **AWS SDKs** — Provides language-specific APIs and takes care of many of the connection details, such as calculating signatures, handling request retries, and handling errors. For more information, see AWS SDKs.
- **HTTPS API** — Provides low-level API actions that you can call using HTTPS requests. For more information, see the AWS Fault Injection Simulator API Reference.

Pricing

You are charged per minute that an action runs, from start to finish. For more information, see AWS FIS Pricing.

Get started with AWS FIS

Complete the following tasks to get set up and start using AWS Fault Injection Simulator (AWS FIS).

Tasks

Plan for experiments

Fault injection is the process of stressing an application in testing or production environments by creating disruptive events, such as server outages or API throttling. From observing how the system responds, you can then implement improvements. When you run experiments on your system, it can help you to identify systemic weaknesses in a controlled manner, before those weaknesses affect the customers who depend on your system. Then you can proactively address the issues to help prevent unpredictable outcomes.

Before you get started running fault injection experiments, we recommend that you familiarize yourself with the following principles and guidelines.

Basic principles and guidelines

Before starting experiments with AWS FIS, take the following steps:

1. **Identify the target deployment for the experiment** — Start by identifying the target deployment. If this is your first experiment, we recommend starting in a pre-production or test environment.
2. **Review the application architecture** — You must ensure that you have identified all of the application components, dependencies, and recovery procedures for each component. Begin with reviewing the application architecture. Depending on the application, refer to the AWS Well-Architected Framework.
3. **Define steady-state behavior** — Define the steady-state behavior of your system in terms of important technical and business metrics, such as latency, CPU load, failed sign-ins per minute, number of retries, or page load speed.
4. **Form a hypothesis** — Form a hypothesis of how you expect the system behavior to change during the experiment. A hypothesis definition follows this format: If *`fault injection action`* is performed, the *`business or technical metric impact`* should not exceed *`value`*. A hypothesis for an authentication service might read as follows: If network latency increases by 10%, there is less than a 1% increase in sign-in failures. After the experiment is completed, you evaluate whether the application resiliency aligns with your business and technical expectations.

We also recommend following these guidelines when working with AWS FIS:

- Always start experimenting with AWS FIS in a test environment. Never start with a production environment. As you progress in your fault injection experiments, you can experiment in other controlled environments beyond the test environment.
- Build your team's confidence in your application resilience by starting with small, simple experiments, such as running the **aws:ec2:stop-instances** action on one target.
- Fault injection can cause real issues. Proceed with caution and make sure that your first fault injections are on test instances so that no customers are affected.

- Test, test, and test some more. Fault injection is meant to be implemented in a controlled environment with well-planned experiments. This allows you to build confidence in the abilities of your application and your tools to withstand turbulent conditions.
- We strongly recommend that you have an excellent monitoring and alerting program in place before you begin. Without it, you won't be able to understand or measure the impact of your experiments, which is critical to sustainable fault injection practices.

Experiment planning guidelines

With AWS FIS, you run experiments on your AWS resources to test your theory of how an application or system will perform under fault conditions.

> **Important**
> AWS FIS carries out real actions on real AWS resources in your system. Therefore, before you get started using AWS FIS to run experiments, we strongly recommend that you first complete a planning phase and a test in a pre-production or test environment.

The following are recommended guidelines for planning your AWS FIS experiments.

- **Review outage history** — Review the previous outages and events for your system. This can help you to build up a picture of the overall health and resiliency of your system. Before you start running experiments on your system, you should address known issues and weaknesses in your system.
- **Identify services with the largest impact** — Review your services and identify the ones that have the biggest impact on your end users or customers if they go down or do not function correctly.
- **Identify the target system** — The target system is the system on which you will run experiments. If you are new to AWS FIS or you have never run fault injection experiments before, we recommend that you start by running experiments on a pre-production or test system.
- **Consult with your team** — Ask what they are worried about. You can form a hypothesis to prove or disprove their concerns. You can also ask your team what they are not worried about. This question can reveal two common fallacies: the sunk cost fallacy and the confirmation bias fallacy. Forming a hypothesis based on your team's answers can help provide more information about the reality of your system's state.
- **Review your application architecture** — Conduct a review of your system or application and ensure that you have identified all of the application components, dependencies, and recovery procedures for every component.

  We recommend that you review the AWS Well-Architected Framework. The framework can help you build secure, high-performing, resilient, and efficient infrastructure for your applications and workloads. For more information, see AWS Well-Architected.
- **Identify the applicable metrics** — You can monitor the impact of an experiment on your AWS resources using Amazon CloudWatch metrics. You can use these metrics to determine the baseline or "steady state" when your application is performing optimally. Then, you can monitor these metrics during or after the experiment to determine the impact. For more information, see Monitor AWS FIS usage metrics with Amazon CloudWatch (p. 40).
- **Define an acceptable performance threshold for your system** — Identify the metric that represents an acceptable, steady state for your system. You will use this metric to create one or more CloudWatch alarms that represent a stop condition for your experiment. If the alarm is triggered, the experiment is automatically stopped. For more information, see Stop conditions for AWS FIS (p. 28).

Set up IAM permissions

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and

authorized (have permissions) to use AWS FIS resources. IAM is an AWS service that you can use with no additional charge.

To use AWS FIS, set up the following permissions.

- Permissions for the IAM users and roles that will work with AWS FIS
- Permissions for AWS FIS that allow it to run experiments on your behalf

Step 1: Set up permissions for IAM users and roles

By default, IAM users do not have permission to work with AWS FIS. You can use IAM identity-based policies to specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied.

The following example policy grants the user full access to the AWS FIS console and API actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FISPermissions",
            "Effect": "Allow",
            "Action": [
                "fis:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ReadOnlyActions",
            "Effect": "Allow",
            "Action": [
                "ssm:Describe*",
                "ssm:Get*",
                "ssm:List*",
                "ec2:DescribeInstances",
                "rds:DescribeDBClusters",
                "ecs:DescribeClusters",
                "ecs:ListContainerInstances",
                "eks:DescribeNodegroup",
                "cloudwatch:DescribeAlarms",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "IAMPassRolePermissions",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/roleName"
        },
        {
            "Sid": "PermissionsToCreateServiceLinkedRole",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "fis.amazonaws.com"
                }
            }
```

```
        }
    ]
}
```

You can modify the preceding policy to restrict permissions to specific API operations. For more policy examples, see AWS Fault Injection Simulator policy examples (p. 53). For more information about identity-based policies, see Identity and access management for AWS Fault Injection Simulator (p. 46).

Step 2: Set up the IAM role for the AWS FIS service

When you create an experiment template, you must specify an IAM role that grants the AWS FIS service permission to perform actions on your behalf. The IAM policy for the IAM role must grant permission to modify the resources that you specify as targets in your experiment template.

The following policy contains the API actions that are needed by AWS FIS to conduct experiments on supported AWS FIS resources. As a best practice, we recommend following the standard security advice of granting least privilege. You can do so by specifying specific resource ARNs or tags in your policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowFISExperimentRoleReadOnly",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ecs:DescribeClusters",
                "ecs:ListContainerInstances",
                "eks:DescribeNodegroup",
                "iam:ListRoles",
                "rds:DescribeDBInstances",
                "rds:DescribeDbClusters",
                "ssm:ListCommands"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowFISExperimentRoleEC2Actions",
            "Effect": "Allow",
            "Action": [
                "ec2:RebootInstances",
                "ec2:StopInstances",
                "ec2:StartInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": "arn:aws:ec2:*:*:instance/*"
        },
        {
            "Sid": "AllowFISExperimentRoleECSActions",
            "Effect": "Allow",
            "Action": [
                "ecs:UpdateContainerInstancesState",
                "ecs:ListContainerInstances"
            ],
            "Resource": "arn:aws:ecs:*:*:container-instance/*"
        },
        {
            "Sid": "AllowFISExperimentRoleEKSActions",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
```

```
            "Resource": "arn:aws:ec2:*:*:instance/*"
        },
        {
            "Sid": "AllowFISExperimentRoleFISActions",
            "Effect": "Allow",
            "Action": [
                "fis:InjectApiInternalError",
                "fis:InjectApiThrottleError",
                "fis:InjectApiUnavailableError"
            ],
            "Resource": "arn:*:fis:*:*:experiment/*"
        },
        {
            "Sid": "AllowFISExperimentRoleRDSReboot",
            "Effect": "Allow",
            "Action": [
                "rds:RebootDBInstance"
            ],
            "Resource": "arn:aws:rds:*:*:db:*"
        },
        {
            "Sid": "AllowFISExperimentRoleRDSFailOver",
            "Effect": "Allow",
            "Action": [
                "rds:FailoverDBCluster"
            ],
            "Resource": "arn:aws:rds:*:*:cluster:*"
        },
        {
            "Sid": "AllowFISExperimentRoleSSMSendCommand",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ssm:*:*:document/*"
            ]
        },
        {
            "Sid": "AllowFISExperimentRoleSSMCancelCommand",
            "Effect": "Allow",
            "Action": [
                "ssm:CancelCommand"
            ],
            "Resource": "*"
        }
    ]
}
```

In addition, the IAM role must have a trust relationship that allows the AWS FIS service to assume the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                  "fis.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
```

```
            "Condition": {}
        }
    ]
}
```

Service-linked roles in AWS FIS

When you start an experiment from an experiment template, AWS FIS creates a service-linked role for you that grants permission to list or describe the resources that you specify as targets in your experiment template. Therefore, the IAM role that grants AWS FIS permission to perform actions on your behalf must have a policy that grants the following:

- Permission to start the experiment
- Permission to create a service-linked role on your behalf

For information about service-linked roles in AWS FIS, see Use service-linked roles for AWS Fault Injection Simulator (p. 59). For information about creating an IAM role for an AWS service, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

Tutorial: Create and run an experiment

To begin using AWS FIS, you can start by setting up and running a simple experiment on test resources. In the following tutorial, you launch two test Amazon EC2 instances. You then create an experiment template that uses the `aws:ec2:stop-instances` action to stop one of the instances, and then stop both instances.

Prerequisites

To complete this tutorial, ensure that you do the following:

- Launch two test EC2 instances in your account. After you launch your instances, take note of the IDs of both instances.
- Ensure that you have created an IAM role that enables the AWS FIS service to perform actions on your behalf. For more information, see Step 2: Set up the IAM role for the AWS FIS service (p. 7).

Step 1: (Recommended) Create a CloudWatch alarm for a stop condition

First, define the steady state for your application or service. The steady state is when your application is performing optimally, defined in terms of business or technical metrics. For example, latency, CPU load, or number of retries. Monitor your application or service to determine the threshold at which its performance is not acceptable. Configure a CloudWatch alarm so that you can stop the experiment if the application or service crosses this threshold.

For more information, see Create a CloudWatch alarm based on a static threshold and Create a CloudWatch alarm based on anomaly detection in the *Amazon CloudWatch User Guide*.

Step 2: Create an experiment template

Create the experiment template using the AWS FIS console. In the template, you specify two actions that will run sequentially for five minutes each. The first action stops one of the test instances, which AWS FIS chooses randomly. The second action stops both test instances.

To create an experiment template

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Choose **Create experiment template**.
4. Enter a description for the template.
5. For **IAM role**, select the IAM role that you created earlier.
6. For **Actions**, choose **Add action**. Complete the following information:
   - For **Name**, enter a name for the action, for example, `StopOneInstance`.
   - For **Action type**, choose **aws:ec2:stop-instances**.
   - For **startInstancesAfterDuration**, enter `5`. This represents a five-minute duration.
7. Choose **Save**.
8. For **Targets**, choose **Edit** for the target that AWS FIS automatically created for you in the previous step. Complete the following steps:
   - For **Name**, replace the default name with a more descriptive name, for example, `OneRandomInstance`.
   - For **Resource type**, ensure that `aws:ec2:instance` is selected.
   - For **Target method**, choose **Resource IDs**, and then choose the IDs of the two test instances that you created earlier.
   - For **Selection mode**, choose **COUNT**. For **Number of resources**, enter `1`.
9. Choose **Save**.
10. Choose **Add target**. Complete the following information:
    - For **Name**, enter a name, for example, `AllInstances`.
    - For **Resource type**, select `aws:ec2:instance`.
    - For **Target method**, choose **Resource IDs**, and then choose the IDs of the two test instances that you created earlier.
    - For **Selection mode**, choose **ALL**.
11. Choose **Save**.
12. Return to the **Actions** section and choose **Add action**. Complete the following information:
    - For **Name**, enter a name for the action, for example, `StopAllInstances`.

      Allowed characters are alphanumeric characters, hyphens (-), and underscores(_). The name must start with a letter. No spaces are allowed. Each action name in the template must be unique.
    - For **Action type**, choose **aws:ec2:stop-instances**.
    - For **Start after**, choose the name of the action that you added earlier (`StopOneInstance`).
    - For **Target**, choose the name of the target that you added in the preceding step (`AllInstances`).
    - For **startInstancesAfterDuration**, enter `5`.
13. Choose **Save**.
14. For **Stop conditions**, select the CloudWatch alarms that you created earlier.
15. (Optional) For **Tags**, choose **Add new tag** and specify a tag key and tag value.
16. Choose **Create experiment template**.

Step 3: Start the experiment

When you have finished creating your experiment template, you can use it to start an experiment.

To start an experiment

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template, and choose **Actions**, **Start experiment**.
4. (Optional) Choose **Actions**, **Manage tags** to add tags for your experiment, and then follow the steps in the console. Then choose **Start experiment**.

Step 4: View the experiment

You can view the progress of a running experiment until the experiment is completed, stopped, or failed.

To view an experiment

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiments**.
3. Select the **Experiment ID** to open the details page for the experiment.
4. To view the state of the experiment, check **State** in the **Details** pane. For more information, see experiment states (p. 38).

Step 5: Clean up

If you no longer need the test EC2 instances that you created for this experiment, you can terminate them.

To terminate an instance using the Amazon EC2 console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the instance, and choose **Actions**, **Instance State**, **Terminate**.
4. Choose **Yes, Terminate** when prompted for confirmation. Repeat the preceding steps for the second instance.

If you no longer need the experiment template, you can delete it.

To delete an experiment template using the AWS FIS console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template, and choose **Actions**, **Delete experiment template**.
4. Enter `delete` when prompted for confirmation, and then choose **Delete experiment template**.

Actions for AWS FIS

An action is the fault injection activity that you run on a target using AWS Fault Injection Simulator (AWS FIS). AWS FIS provides preconfigured actions for specific types of targets across AWS services. You add actions to an experiment template, which you then use to run experiments.

Contents

Action identifiers

Each AWS FIS action has an identifier with the following format:

```
aws:service-name:action-type
```

For example, the aws:ec2:stop-instances (p. 15) action stops the target Amazon EC2 instances.

Action parameters

Some AWS FIS actions have additional parameters that are specific to the action. These parameters are used to pass information to AWS FIS when the action is run.

AWS FIS supports custom fault types using the `aws:ssm:send-command` action, which uses the SSM Agent and an SSM command document to create the fault condition on the targeted instances. The `aws:ssm:send-command` action includes a `documentArn` parameter that takes the Amazon Resource Name (ARN) of an SSM document as a value. You specify values for parameters when you add the action to your experiment template.

For more information about specifying parameters for the `aws:ssm:send-command` action, see Use the aws:ssm:send-command action (p. 19).

Where possible, you can input a rollback configuration (also referred to as a *post action*) within the action parameters. A post action returns the target to the state that it was in before the action was run. The post action runs after the time specified in the action duration. Not all actions can support post actions. For example, if the action terminates an Amazon EC2 instance, you cannot recover the instance after it has been terminated.

AWS FIS actions reference

This reference describes the common actions in AWS FIS, including information about the action parameters and the required IAM permissions. You can also list the supported AWS FIS actions using the AWS FIS console or the list-actions command from the AWS Command Line Interface (AWS CLI).

For more information, see Actions for AWS FIS (p. 12) and How AWS Fault Injection Simulator works with IAM (p. 50).

Actions

Fault injection actions

AWS FIS supports the following fault injection actions.

Actions

aws:fis:inject-api-internal-error

Runs the AWS FIS action InjectApiInternalError on the target IAM role.

Resource type

- **aws:iam:role**

Parameters

- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.
- **service** – The target AWS API namespace. The supported value is `ec2`.
- **percentage** – The percentage (1-100) of calls to inject the fault into.
- **operations** – The operations to inject the fault into, separated using commas. For a list of the API actions for the `ec2` namespace, see Actions in the *Amazon EC2 API Reference*.

Permissions

- fis:InjectApiInternalError

aws:fis:inject-api-throttle-error

Runs the AWS FIS action InjectApiThrottleError on the target IAM role.

Resource type

- **aws:iam:role**

Parameters

- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.

- **service** – The target AWS API namespace. The supported value is `ec2`.
- **percentage** – The percentage (1-100) of calls to inject the fault into.
- **operations** – The operations to inject the fault into, separated using commas. For a list of the API actions for the `ec2` namespace, see Actions in the *Amazon EC2 API Reference*.

Permissions

- fis:InjectApiThrottleError

aws:fis:inject-api-unavailable-error

Runs the AWS FIS action InjectApiUnavailableError on the target IAM role.

Resource type

- **aws:iam:role**

Parameters

- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.
- **service** – The target AWS API namespace. The supported value is `ec2`.
- **percentage** – The percentage (1-100) of calls to inject the fault into.
- **operations** – The operations to inject the fault into, separated using commas. For a list of the API actions for the `ec2` namespace, see Actions in the *Amazon EC2 API Reference*.

Permissions

- fis:InjectApiUnavailableError

Wait action

AWS FIS supports the following wait action.

aws:fis:wait

Runs the AWS FIS wait action.

Parameters

- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.

Permissions

- None

Amazon EC2 actions

AWS FIS supports the following Amazon EC2 actions.

Actions

aws:ec2:reboot-instances

Runs the Amazon EC2 API action RebootInstances on the target EC2 instances.

Resource type

- **aws:ec2:instance**

Parameters

- None

Permissions

- ec2:RebootInstances

aws:ec2:stop-instances

Runs the Amazon EC2 API action StopInstances on the target EC2 instances.

Resource type

- **aws:ec2:instance**

Parameters

- **startInstancesAfterDuration** – Optional. The time to wait before starting the instance. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.

Permissions

- ec2:StopInstances
- ec2:StartInstances

aws:ec2:terminate-instances

Runs the Amazon EC2 API action TerminateInstances on the target EC2 instances.

Resource type

- **aws:ec2:instance**

Parameters

- None

Permissions

- ec2:TerminateInstances

Amazon ECS actions

AWS FIS supports the following Amazon ECS actions.

aws:ecs:drain-container-instances

Runs the Amazon ECS API action UpdateContainerInstancesState to drain the specified percentage of underlying Amazon EC2 instances on the target clusters.

Resource type

- **aws:ecs:cluster**

Parameters

- **drainagePercentage** – The percentage (1-100).
- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.

Permissions

- ecs:DescribeClusterInstances
- ecs:UpdateContainerInstancesState
- ecs:ListContainerInstances

Amazon EKS actions

AWS FIS supports the following Amazon EKS actions.

aws:eks:terminate-nodegroup-instance

Runs the Amazon EC2 API action TerminateInstances on the target node group.

Resource type

- **aws:eks:nodegroup**

Parameters

- **InstanceTerminationPercentage** – The percentage (1-100) of instances to terminate.

Permissions

- ec2:DescribeInstances
- ec2:TerminateInstances

Amazon RDS actions

AWS FIS supports the following Amazon RDS actions.

Actions

aws:rds:failover-db-cluster

Runs the Amazon RDS API action FailoverDBCluster on the target Aurora DB cluster.

Resource type

- **aws:rds:cluster**

Parameters

- None

Permissions

- rds:FailoverDBCluster

aws:rds:reboot-db-instances

Runs the Amazon RDS API action RebootDBInstance on the target DB instance.

Resource type

- **aws:rds:db**

Parameters

- **forceFailover** – Optional. If the value is true, and if instances are Multi-AZ, forces failover from one Availability Zone to another. The default is false.

Permissions

- rds:RebootDBInstance

Systems Manager actions

AWS FIS supports the following Systems Manager actions.

Actions

aws:ssm:send-command

Runs the Systems Manager API action SendCommand on the target EC2 instances. For more information, see Use the aws:ssm:send-command action (p. 19).

Resource type

- **aws:ec2:instance**

Parameters

- **documentArn** – The Amazon Resource Name (ARN) of the document. No default value.
- **documentVersion** – Optional. The version of the document. If empty, the default version runs.
- **documentParameters** – Conditional. Any parameters specific to the document. With the AWS CLI, format as an escaped JSON string. With the AWS FIS console, you can use a JSON string without escaping.
- **duration** – The duration. With the AWS FIS API, the value is a string in ISO 8601 format. For example, PT1M represents one minute. With the AWS FIS console, you enter the number of minutes.

Permissions

- ssm:SendCommand
- ssm:ListCommands
- ssm:CancelCommand

aws:ssm:start-automation-execution

Runs the Systems Manager API action StartAutomationExecution.

Resource type

- None.

Parameters

- **documentArn** – The Amazon Resource Name (ARN) of the automation document. No default value.
- **documentVersion** – Optional. The version of the document. If empty, the default version runs.
- **documentParameters** – Conditional. Any parameters specific to the document. With the AWS CLI, format as an escaped JSON string. With the AWS FIS console, you can use a JSON string without escaping.
- **maxDuration** – The maximum time allowed for the automation execution to complete.

Permissions

- ssm:GetAutomationExecution
- ssm:StartAutomationExecution
- ssm:StopAutomationExecution
- iam:PassRole (if the automation assumes a role)

Use Systems Manager SSM documents with AWS FIS

AWS FIS supports custom fault types through the AWS Systems Manager SSM Agent and the AWS FIS action aws:ssm:send-command (p. 18). Pre-configured Systems Manager SSM documents (SSM documents) that can be used to create common fault injection actions are available as public AWS documents that begin with the AWSFIS- prefix.

SSM Agent is Amazon software that can be installed and configured on Amazon EC2 instances, on-premises servers, or virtual machines (VMs). This makes it possible for Systems Manager to manage these resources. The agent processes requests from Systems Manager, and then runs them as specified in the request. You can include your own SSM document to inject custom faults, or reference one of the public Amazon-owned documents.

Requirements

For actions that require SSM Agent to run the action on the target, you must ensure the following:

- The agent is installed on the target. SSM Agent is installed by default on some Amazon Machine Images (AMIs). Otherwise, you can install the SSM Agent on your instances. For more information, see Manually install SSM Agent for EC2 instances in the *AWS Systems Manager User Guide*.
- Systems Manager has permission to perform actions on your instances. You grant access using an IAM instance profile. For more information, see Create an IAM instance profile for Systems Manager and Attach an IAM instance profile to an EC2 instance in the *AWS Systems Manager User Guide*.

Use the aws:ssm:send-command action

An SSM document defines the actions that Systems Manager performs on your managed instances. Systems Manager includes a number of pre-configured documents, or you can create your own. For more information about creating your own SSM document, see Creating Systems Manager documents in the *AWS Systems Manager User Guide*. For more information about SSM documents in general, see AWS Systems Manager documents in the *AWS Systems Manager User Guide*.

AWS FIS provides pre-configured SSM documents. You can view the pre-configured SSM documents under **Documents** in the AWS Systems Manager console: https://console.aws.amazon.com/systems-manager/documents. You can also choose from a selection of pre-configured documents in the AWS FIS console. For more information, see Pre-configured AWS FIS SSM documents (p. 20).

To use an SSM document in your AWS FIS experiments, you can use the aws:ssm:send-command (p. 18) action. This action fetches and runs the specified SSM document on your target instances.

When you use the `aws:ssm:send-command` action in your experiment template, you must specify additional parameters for the action, including the following:

- **documentArn** – Required. The Amazon Resource Name (ARN) of the SSM document.
- **documentParameters** – Conditional. Any parameters that are specific to the SSM document. Not all documents have required parameters. With the AWS CLI, you specify these parameters and their values as an escaped JSON string. With the AWS FIS console, you can use a JSON string without escaping.
- **documentVersion** – Optional. The version of the SSM document to run.

You can view the information for an SSM document (including the parameters for the document) by using the Systems Manager console or the command line.

To view information about an SSM document using the console

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Documents**.
3. Select the document, and choose the **Details** tab.

To view information about an SSM document using the command line

Use the SSM describe-document command.

Pre-configured AWS FIS SSM documents

You can use pre-configured AWS FIS SSM documents with the `aws:ssm:send-command` action in your experiment templates.

Requirement

The pre-configured SSM documents provided by AWS FIS are supported only on Amazon Linux and Ubuntu. On other Linux systems and Windows, you can use the `aws:ssm:send-command` action to run your own SSM document.

Actions

When you create your experiment template, select one of the following actions to use the corresponding pre-configured SSM document.

- aws:ssm:send-command/AWSFIS-Run-CPU-Stress (p. 20)
- aws:ssm:send-command/AWSFIS-Run-Kill-Process (p. 20)
- aws:ssm:send-command/AWSFIS-Run-Memory-Stress (p. 21)
- aws:ssm:send-command/AWSFIS-Run-Network-Latency (p. 21)

AWSFIS-Run-CPU-Stress

Runs CPU stress on an instance using the **stress-ng** tool. Uses the AWSFIS-Run-CPU-Stress SSM document and the following document parameters:

- **DurationSeconds** – Required. The duration of the CPU stress test, in seconds.
- **CPU** – Optional. The number of CPU stressors to use. The default is 0, which uses all CPU stressors.
- **InstallDependencies** – Optional. If the value is `true`, Systems Manager installs the required dependencies on the target instances if they are not already installed. The default is `true`. The dependency is **stress-ng**.

The following is an example.

```
{"DurationSeconds":"60", "InstallDependencies":"True"}
```

AWSFIS-Run-Kill-Process

Stops the specified process in the instance, using the **killall** command. Uses the AWSFIS-Run-Kill-Process SSM document with the following document parameters:

- **ProcessName** – Required. The name of the process to stop.
- **Signal** – Optional. The signal to send along with the command. The possible values are `SIGTERM` (which the receiver can choose to ignore) and `SIGKILL` (which cannot be ignored). The default is `SIGTERM`.

The following is an example.

```
{"ProcessName":"myapplication", "Signal":"SIGTERM"}
```

AWSFIS-Run-Memory-Stress

Runs memory stress on an instance using the **stress-ng** tool. Uses the AWSFIS-Run-Memory-Stress SSM document with the following document parameters:

- **DurationSeconds** – Required. The duration of the memory stress test, in seconds.
- **Workers** – Optional. The number of virtual memory stressors. The default is 1.
- **Percent** – Required. The percentage of virtual memory to use during the memory stress test.
- **InstallDependencies** – Optional. If the value is `true`, Systems Manager installs the required dependencies on the target instances if they are not already installed. The default is `true`. The dependency is **stress-ng**.

The following is an example.

```
{"Percent":"80", "DurationSeconds":"60", "InstallDependencies":"True"}
```

AWSFIS-Run-Network-Latency

Adds latency to the network interface using the **tc** tool. Uses the AWSFIS-Run-Network-Latency SSM document with the following document parameters:

- **Interface** – Optional. The network interface. The default is `eth0`.
- **DelayMilliseconds** – Optional. The delay, in milliseconds. The default is 200.
- **DurationSeconds** – Required. The duration of the network latency test, in seconds.
- **InstallDependencies** – Optional. If the value is `true`, Systems Manager installs the required dependencies on the target instances if they are not already installed. The default is `true`. The dependencies are **tc** and **atd**.

The following is an example.

```
{"DelayMilliseconds":"200", "Interface":"eth0", "DurationSeconds":"60",
 "InstallDependencies":"True"}
```

Experiment templates for AWS FIS

An experiment template contains one or more actions to run on specified targets during an experiment. It also contains the stop conditions that prevent the experiment from going out of bounds. After you create an experiment template, you can use it to run an experiment.

Template components

You'll use the following are the components to construct experiment templates. To create an experiment template using the AWS Management Console, see Create an experiment template (p. 29). To create an experiment template using the AWS CLI, see Example AWS FIS experiment templates (p. 32).

Action set

An action set contains the AWS FIS actions (p. 12) that you want to run. You must specify at least one action set in your experiment template. Actions can be run in a set order that you specify, or they can be run simultaneously. For more information, see Action set for AWS FIS (p. 22).

Some AWS FIS actions require the use of the AWS Systems Manager Agent (SSM Agent) on the target resource. For more information, see Use Systems Manager SSM documents with AWS FIS (p. 19).

You have quotas on the number of actions and the action duration that you can specify in a template. For more information, see AWS Fault Injection Simulator endpoints and quotas in the *AWS General Reference*.

Targets

One or more AWS resources on which a specific action is carried out. For more information, see Targets for AWS FIS (p. 23).

IAM role

The ARN of an IAM role that grants the AWS FIS service permission to carry out actions on your behalf. For more information about setting up this role, see Set up IAM permissions (p. 5).

Stop conditions

One or more CloudWatch alarms. If a stop condition (p. 28) is triggered while an experiment is running, AWS FIS stops the experiment.

Action set for AWS FIS

To create an experiment template, you must define one or more actions to make up the action set. For a list of predefined actions provided by AWS FIS, see Actions (p. 12).

An action runs on the target resources that you specify. For more information, see Targets (p. 23).

When you define an action, you do the following:

- Choose the AWS FIS action to use
- Name the action
- Specify any applicable action parameters

- Specify when to start the action during an experiment
- Name the target

By default, an action starts at the beginning of the experiment. You can specify that the action must start after one or more other actions are complete.

An action is considered complete when the specified duration has elapsed. If you specify a duration of 5 minutes, AWS FIS considers the action complete after 5 minutes. It then starts the next action, or completes the experiment if all other actions are complete.

Duration can be the length of time that an action condition is maintained or the length of time for which metrics are monitored. For example, latency is injected for the duration of time specified. For near instantaneous action types, such as terminating an instance, a longer duration will continue to monitor stop conditions for the duration of time specified.

If an action includes a post action within the action parameters, the post action runs at the end of the specified action duration. The time it takes to complete the post action might cause a delay between the specified action duration and the beginning of the next action (or the end of the experiment, if all other actions are complete).

You cannot loop an action. An action can only be run once during an experiment. To run the same AWS FIS action more than once in the same experiment, specify a new action in the template (with a different name), and use the same AWS FIS action.

Targets for AWS FIS

A target is one or more AWS resources on which an action is performed by AWS Fault Injection Simulator (AWS FIS) during an experiment. You define targets when you create an experiment template (p. 29). You can use the same target for multiple actions in your experiment template.

When you define a target, you specify the following:

- The resource type
- How to identify the resources (through resource IDs, filters, or tags)
- Which of the identified resources to run the action on (the selection mode)

Contents

Resource types

Each AWS FIS action is performed on a specific AWS resource type. When you define a target, you must specify exactly one resource type. When you specify a target for an action, the target must be the resource type supported by the action.

The following resource types are supported by AWS FIS:

- **aws:ec2:instance** – Amazon EC2 instances
- **aws:ecs:cluster** – Amazon ECS clusters

- **aws:eks:nodegroup** – Amazon EKS node groups
- **aws:iam:role** – IAM roles
- **aws:rds:cluster** – Amazon Aurora DB cluster
- **aws:rds:db** – Amazon RDS DB instances

Identify target resources

When you define a target in the AWS FIS console, you can choose specific AWS resources (of a specific resource type) to target in your account. Or, you can let AWS FIS identify a group of resources based on the criteria that you provide.

To identify your target resources, you can specify the following:

- **Resource IDs** – The resource IDs of specific AWS resources. For example, the resource ID of an Amazon EC2 instance, such as `i-1122334455aabbccd`. All resource IDs must be the same resource type.
- **Resource filters** – The path and values that represent resources with specific attributes. For more information, see Resource filters (p. 24).
- **Resource tags** – The tags applied to target resources. For example, you can specify that the target EC2 instances must include the tag env=test.

You must specify at least one resource ID or at least one resource tag for the target. You cannot specify both a resource ID and a resource tag for the same target. You also cannot specify resource IDs and resource filters in the same target.

Resource filters

Resource filters are queries that identify target resources according to specific attributes. AWS FIS applies the query to the output of an API action that contains the canonical description of the AWS resource, according to the resource type that you specify. Resources that have attributes that match the query are included in the target definition.

Each filter is expressed as an attribute path and possible values. A path is a sequence of elements, separated by periods, the describe the path to reach an attribute in the output of the **Describe** action for a resource.

```
"filters": [
    {
        "path": "component.component.component",
        "values": [
            "string"
        ]
    }
],
```

The following table includes the API actions and AWS CLI commands that you can use to get the canonical descriptions for each resource type. AWS FIS runs these actions on your behalf to apply the filters that you specify. The corresponding documentation describes the resources that are included in the results by default. For example, the documentation for **DescribeInstances** states that recently terminated instances might appear in the results.

Resource type	API action	AWS CLI command
aws:ec2:instance	DescribeInstances	describe-instances

Resource type	API action	AWS CLI command
aws:ecs:cluster	DescribeClusters	describe-clusters
aws:eks:nodegroup	DescribeNodegroup	describe-nodegroup
aws:iam:role	ListRoles	list-roles
aws:rds:cluster	DescribeDBClusters	describe-db-clusters
aws:rds:db	DescribeDBInstances	describe-db-instances

The following logic applies to all resource filters:

- Values inside a filter – `OR`
- Values across filters – `AND`

Example: EC2 instances

When you specify a filter for an action that supports the **aws:ec2:instance** resource type, AWS FIS uses the Amazon EC2 **describe-instances** command in your account and applies the filter to identify the targets.

The **describe-instances** command returns JSON output where each instance is a structure under `Instances`. The following is partial output that includes fields marked with *`italics`*. We'll provide examples that use these fields to specify an attribute path from the structure of the JSON output.

```
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "ImageId": "ami-00111111111111111",
                    "InstanceId": "i-00aaaaaaaaaaaaaaa",
                    "InstanceType": "t2.micro",
                    "KeyName": "virginia-kp",
                    "LaunchTime": "2020-09-30T11:38:17.000Z",
                    "Monitoring": {
                        "State": "disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-10-0-1-240.ec2.internal",
                    "PrivateIpAddress": "10.0.1.240",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-203-0-113-17.compute-1.amazonaws.com",
                    "PublicIpAddress": "203.0.113.17",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    },
                    "StateTransitionReason": "",
                    "SubnetId": "subnet-aabbcc11223344556",
                    "VpcId": "vpc-00bbbbbbbbbbbbbbbbb",
                    ...
                },
                ...
```

```
                {
                    ...
                }
            ],
            "OwnerId": "123456789012",
            "ReservationId": "r-aaaaaabbbbb111111"
        },
        ...
    ]
}
```

To select instances in a specific Availability Zone using a resource filter, specify the attribute path for `AvailabilityZone` and the code for the Availability Zone as the value. For example:

```
"filters": [
    {
        "path": "Placement.AvailabilityZone",
        "values": [ "us-east-1a" ]
    }
],
```

To select instances in a specific subnet using a resource filter, specify the attribute path for `SubnetId` and the ID of the subnet as the value. For example:

```
"filters": [
    {
        "path": "SubnetId",
        "values": [ "subnet-aabbcc11223344556" ]
    }
],
```

To select instances that are in a specific instance state, specify the attribute path for `Name` and one of the following state names as the value: `pending` | `running` | `shutting-down` | `terminated` | `stopping` | `stopped`. For example:

```
"filters": [
    {
        "path": "State.Name",
        "values": [ "running" ]
    }
],
```

Example: Amazon RDS cluster (DB cluster)

When you specify a filter for an action that supports the **aws:rds:cluster** resource type, AWS FIS runs the Amazon RDS **describe-db-clusters** command in your account and applies the filter to identify the targets.

The **describe-db-clusters** command returns JSON output similar to the following for each DB cluster. The following is partial output that includes fields marked with *italics*. We'll provide examples that use these fields to specify an attribute path from the structure of the JSON output.

```
[
    {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-2a",
            "us-east-2b",
            "us-east-2c"
        ],
```

```
        "BackupRetentionPeriod": 7,
        "DatabaseName": "",
        "DBClusterIdentifier": "database-1",
        "DBClusterParameterGroup": "default.aurora-postgresql11",
        "DBSubnetGroup": "default-vpc-01234567abc123456",
        "Status": "available",
        "EarliestRestorableTime": "2020-11-13T15:08:32.211Z",
        "Endpoint": "database-1.cluster-example.us-east-2.rds.amazonaws.com",
        "ReaderEndpoint": "database-1.cluster-ro-example.us-east-2.rds.amazonaws.com",
        "MultiAZ": false,
        "Engine": "aurora-postgresql",
        "EngineVersion": "11.7",
        ...
    }
]
```

To apply a resource filter that returns only the DB clusters that use a specific DB engine, specify the attribute path as `Engine` and the value as `aurora-postgresql` as shown in the following example.

```
"filters": [
    {
        "path": "Engine",
        "values": [ "aurora-postgresql" ]
    }
],
```

To apply a resource filter that returns only the DB clusters in a specific Availability Zone, specify the attribute path and value as shown in the following example.

```
"filters": [
    {
        "path": "AvailabilityZones",
        "values": [ "us-east-2a" ]
    }
],
```

Selection mode

By default, AWS FIS runs an action on all of the targets that are identified by the resource IDs, filters, or tags that you provide. Alternatively, you can scope the identified resources as follows:

- COUNT(n) – Run the action on the specified number of targets, chosen from the identified targets at random. For example, COUNT(1) selects one of the identified targets.
- PERCENT(n) – Run the action on the specified percentage of targets, chosen from the identified targets at random. For example, PERCENT(25) selects 25% of the identified targets.

If you have an odd number of resources and specify 50%, AWS FIS rounds down. For example, if you add five Amazon EC2 instances as targets and scope to 50%, AWS FIS rounds down to two instances. You can't specify a percentage that is less than one resource. For example, if you add four Amazon EC2 instances and scope to 5%, AWS FIS can't select an instance.

Regardless of which selection mode you use, if the scope that you specify identifies no resources, the experiment fails.

Example target

The following is an example of a target:

- **Name**: RandomTestInstance
- **Resource type**: `aws:ec2:instance`
- **Filter**: **Attribute path**: `VpcId`, **Value**: `vpc-aabbcc11223344556`
- **Resource tag**: `env` = `prod`
- **Selection mode**: COUNT (1)

The resources for this target are Amazon EC2 instances in the specified VPC with the tag env=prod. The selection mode specifies that one instance that meets this criteria is chosen at random.

```
{
    "RandomTestInstance": {
        "resourceType": "aws:ec2:instance",
        "resourceTags": {
            "env": "prod"
        },
        "filters": [
            {
                "path": "VpcId",
                "values": [
                    "vpc-aabbcc11223344556"
                ]
            }
        ],
        "selectionMode": "COUNT(1)"
    }
}
```

Stop conditions for AWS FIS

AWS Fault Injection Simulator (AWS FIS) provides controls and guardrails for you to run experiments safely on AWS workloads. A *stop condition* is a mechanism to stop an experiment if it reaches a threshold that you define as a Amazon CloudWatch alarm. If a stop condition is triggered during an experiment, AWS FIS stops the experiment, and the experiment enters the **stopping** state. You cannot resume a stopped experiment.

To create a stop condition, first define the steady state for your application or service. The steady state is when your application is performing optimally, defined in terms of business or technical metrics. For example, latency, CPU load, or number of retries. You can use the steady state to create a CloudWatch alarm that you can use to stop an experiment if your application or service reaches a state where its performance is not acceptable.

When you create an experiment template, you specify one or more stop conditions by specifying the CloudWatch alarms that you created. Your account has a quota on the number of stop conditions that you can specify in an experiment template. For more information, see Quotas for AWS Fault Injection Simulator (p. 67).

For more information about creating CloudWatch alarms, see Create a CloudWatch alarm based on a static threshold and Create a CloudWatch alarm based on anomaly detection in the *Amazon CloudWatch User Guide*.

For more information about the CloudWatch metrics that are available for the resource types supported by AWS FIS, see the following:

- Monitor your instances using CloudWatch
- Amazon ECS CloudWatch metrics

- Overview of monitoring Amazon RDS

Work with AWS FIS experiment templates

You can create and manage experiment templates using the AWS FIS console or the command line After you create an experiment template, you can use it to run an experiment.

Tasks

Create an experiment template

Before you begin, ensure that you have reviewed the components (p. 22) that you need to create the template.

To create an experiment template using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Choose **Create experiment template**.
4. Enter a description for the template.
5. For **IAM role**, select an IAM role that grants the AWS FIS service permission to perform actions on your behalf. For more information, see Set up IAM permissions (p. 5).
6. For **Actions**, specify the set of actions for the template. For each action, choose **Add action** and complete the following:

   - For **Name**, enter a name for the action.

     Allowed characters are alphanumeric characters, hyphens (-), and underscores(_). The name must start with a letter. No spaces are allowed. Each action name in the template must be unique.
   - (Optional) For **Description**, enter a description for the action. Up to 512 characters are allowed.
   - For **Action type**, select the action.
   - (Optional) For **Start after**, select another action in the template that must be completed before the current action starts. The other action must already be specified in your template. To run the action at the start of the experiment, leave the default value. For more information, see Action set for AWS FIS (p. 22).
   - For **Target**, select a target that you defined in the **Targets** section. If you have not defined a target for this action yet, AWS FIS creates a new target for you.
   - For **Duration**, enter the number of minutes to run the action.
   - (Optional) For **Action parameters**, specify any additional parameter values for the action. Fields for additional parameter values are only available if the selected AWS FIS action has them.
   - Choose **Save**.
7. For **Targets**, define the target resources on which to carry out the actions. Choose **Edit** to edit the target that AWS FIS created for you (step 6), or choose **Add target**. For each target, do the following:

- For **Name**, enter a name for the target.

  Allowed characters are alphanumeric characters, hyphens (-), and underscores(_). The name must start with a letter. No spaces are allowed. Each target name in the template must be unique.
- For **Resource type**, select a supported resource type for the action.
- For **Target method**, select one of the following:
  - **Resource IDs**: Specify one or more Resource IDs.
  - **Resource tags and filters**: For **Resource filters**, specify one or more filters to apply to the output of an API action that describes the resource type. For more information, see Resource filters (p. 24). For **Resource tags**, specify the tags of the target resources.

    Note
    You must specify at least one resource ID or at least one resource tag for a target. For more information about the target options, see Targets for AWS FIS (p. 23).
- For **Selection mode**, choose whether to narrow down the identified resources to a specific count at random, or a percentage. By default, all identified resources are included in the target.
- Choose **Save**.

8. If required, return to the **Actions** section, and update an action with the target that you created. Choose **Edit**, and for **Target**, select the target name. You can use the same target for multiple actions.
9. For **Stop conditions**, select one or more Amazon CloudWatch alarms for the stop conditions. For more information, see Stop conditions for AWS FIS (p. 28).
10. (Optional) For **Tags**, choose **Add new tag** and specify a tag key and tag value. Tags you add are applied to your experiment template, not the experiments that are run using the template.
11. Choose **Create experiment template**.

To create an experiment template using the CLI

Use the create-experiment-template command.

You can load an experiment template from a JSON file.

Use the `--cli-input-json` parameter.

```
aws fis create-experiment-template --cli-input-json fileb://<path-to-json-file>
```

For more information, see Generating a CLI skeleton template in the *AWS Command Line Interface User Guide*. For example templates, see Example AWS FIS experiment templates (p. 32).

View experiment templates

You can view the experiment templates that you created.

To view an experiment template using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. To view information about a specific template, select the **Experiment template ID**.
4. In the **Details** section, you can view the description and stop conditions for the template.
5. To view the actions for the experiment template, choose **Actions**.
6. To view the targets for the experiment template, choose **Targets**.
7. To view the tags for the experiment template, choose **Tags**.

To view an experiment template using the CLI

Use the list-experiment-templates command to get a list of experiment templates, and use the get-experiment-template command to get information about a specific experiment template.

Start an experiment from a template

After you have created an experiment template, you can start experiments using that template.

When you start an experiment, we create a snapshot of the specified template and use that snapshot to run the experiment. Therefore, if the experiment template is updated or deleted while the experiment is running, those changes have no impact on the running experiment.

> **Note**
> When you start an experiment, AWS FIS creates a service-linked role on your behalf. For more information, see Use service-linked roles for AWS Fault Injection Simulator (p. 59).

After you start the experiment, you can stop it at any time. For more information, see Stop an experiment (p. 39).

To start an experiment using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template, and choose **Actions**, **Start experiment**.
4. (Optional) To add a tag, choose **Add new tag**, and specify a key and value. To remove a tag, choose **Remove**.
5. Choose **Start experiment**. When prompted, enter `start` and choose **Start experiment**.

To start an experiment using the CLI

Use the start-experiment command.

Update an experiment template

You can update an existing experiment template. When you update an experiment template, the changes do not affect any running experiments that use the template.

To update an experiment template using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template, and choose **Actions**, **Update experiment template**.
4. Modify the template details as needed, and choose **Update experiment template**.

To update an experiment template using the CLI

Use the update-experiment-template command.

Tag experiment templates

You can apply your own tags to experiment templates to help you organize them. You can also implement tag-based IAM policies (p. 52) to control access to experiment templates.

To tag an experiment template using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template and choose **Actions**, **Manage tags**.
4. To add a new tag, choose **Add new tag**, and then specify a key and value.

   To remove a tag, choose **Remove** for the tag.
5. Choose **Save**.

To tag an experiment template using the CLI

Use the tag-resource command.

Delete an experiment template

If you no longer need an experiment template, you can delete it. When you delete an experiment template, any running experiments that use the template are not affected. The experiment continues to run until completed or stopped. However, experiment templates that are deleted are not available for viewing from the **Experiments** page in the console.

To delete an experiment template using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiment templates**.
3. Select the experiment template, and choose **Actions**, **Delete experiment template**.
4. Enter `delete` when prompted for confirmation, and then choose **Delete experiment template**.

To delete an experiment template using the CLI

Use the delete-experiment-template command.

Example AWS FIS experiment templates

If you're using the AWS FIS API or a command line tool to create an experiment template, you can construct the template in JavaScript Object Notation (JSON). For more information about the components of an experiment template, see Template components (p. 22).

To create an experiment using one of the example templates, save it to a JSON file (for example, `my-template.json`), replace the placeholder values in *`italics`* with your own values, and then run the following create-experiment-template command.

```
aws fis create-experiment-template --cli-input-json file://my-template.json
```

Example templates

- Stop EC2 instances based on filters (p. 33)
- Stop a specified number of EC2 instances (p. 33)
- Run a pre-configured AWS FIS SSM document (p. 34)
- Run a predefined Automation runbook (p. 35)
- Throttle API actions on EC2 instances with the target IAM role (p. 35)

Stop EC2 instances based on filters

The following example stops all running Amazon EC2 instances in the specified Region with the specified tag in the specified VPC. It restarts them after two minutes.

```
{
    "tags": {
        "Name": "StopEC2InstancesWithFilters"
    },
    "description": "Stop and restart all instances in us-east-1b with the tag env=prod in
 the specified VPC",
    "targets": {
        "myInstances": {
            "resourceType": "aws:ec2:instance",
            "resourceTags": {
                "env": "prod"
            },
            "filters": [
                {
                    "path": "Placement.AvailabilityZone",
                    "values": ["us-east-1b"]
                },
                {
                    "path": "State.Name",
                    "values": ["running"]
                },
                {
                    "path": "VpcId",
                    "values": [ "vpc-aabbcc11223344556"]
                }
            ],
            "selectionMode": "ALL"
        }
    },
    "actions": {
        "StopInstances": {
            "actionId": "aws:ec2:stop-instances",
            "description": "stop the instances",
            "parameters": {
                "startInstancesAfterDuration": "PT2M"
            },
            "targets": {
                "Instances": "myInstances"
            }
        }
    },
    "stopConditions": [
        {
            "source": "aws:cloudwatch:alarm",
            "value": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name"
}
```

Stop a specified number of EC2 instances

The following example stops three instances with the specified tag. AWS FIS selects the specific instances to stop at random. It restarts these instances after two minutes.

```
{
```

```
    "tags": {
        "Name": "StopEC2InstancesByCount"
    },
    "description": "Stop and restart three instances with the specified tag",
    "targets": {
        "myInstances": {
            "resourceType": "aws:ec2:instance",
            "resourceTags": {
                "env": "prod"
            },
            "selectionMode": "COUNT(3)"
        }
    },
    "actions": {
        "StopInstances": {
            "actionId": "aws:ec2:stop-instances",
            "description": "stop the instances",
            "parameters": {
                "startInstancesAfterDuration": "PT2M"
            },
            "targets": {
                "Instances": "myInstances"
            }
        }
    },
    "stopConditions": [
        {
            "source": "aws:cloudwatch:alarm",
            "value": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name"
}
```

Run a pre-configured AWS FIS SSM document

The following example runs a CPU fault injection for 60 seconds on the specified EC2 instance using a pre-configured AWS FIS SSM document, AWSFIS-Run-CPU-Stress (p. 20). AWS FIS monitors the experiment for two minutes.

```
{
    "tags": {
        "Name": "CPUStress"
    },
    "description": "Run a CPU fault injection on the specified instance",
    "targets": {
        "myInstance": {
            "resourceType": "aws:ec2:instance",
            "resourceArns": ["arn:aws:ec2:us-east-1:111122223333:instance/instance-id"],
            "selectionMode": "ALL"
        }
    },
    "actions": {
        "CPUStress": {
            "actionId": "aws:ssm:send-command",
            "description": "run cpu stress using ssm",
            "parameters": {
                "duration": "PT2M",
                "documentArn": "arn:aws:ssm:us-east-1::document/AWSFIS-Run-CPU-Stress",
                "documentParameters": "{\"DurationSeconds\": \"60\", \"InstallDependencies
\": \"True\", \"CPU\": \"0\"}"
            },
            "targets": {
```

```
                "Instances": "myInstance"
            }
        }
    },
    "stopConditions": [
        {
            "source": "aws:cloudwatch:alarm",
            "value": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name"
}
```

Run a predefined Automation runbook

The following example publishes a notification to Amazon SNS using a runbook provided by Systems Manager, AWS-PublishSNSNotification. The role must have permissions to publish notifications to the specified SNS topic.

```
{
    "description": "Publish event through SNS",
    "stopConditions": [
        {
            "source": "none"
        }
    ],
    "targets": {
    },
    "actions": {
        "sendToSns": {
            "actionId": "aws:ssm:start-automation-execution",
            "description": "Publish message to SNS",
            "parameters": {
                "documentArn": "arn:aws:ssm:us-east-1::document/AWS-
PublishSNSNotification",
                "documentParameters": "{\"Message\": \"Hello, world\", \"TopicArn\":
 \"arn:aws:sns:us-east-1:111122223333:topic-name\"}",
                "maxDuration": "PT1M"
            },
            "targets": {
            }
        }
    },
    "roleArn": "arn:aws:iam::111122223333:role/role-name"
}
```

Throttle API actions on EC2 instances with the target IAM role

The following example throttles 100% percent of the calls to the specified API actions on EC2 instances with the specified IAM role.

```
{
    "tags": {
        "Name": "ThrottleEC2APIActions"
    },
    "description": "Throttle the specified EC2 API actions on the specified IAM role",
    "targets": {
        "myRole": {
```

```
            "resourceType": "aws:iam:role",
            "resourceArns": ["arn:aws:iam::111122223333:role/role-name"],
            "selectionMode": "ALL"
        }
    },
    "actions": {
        "ThrottleAPI": {
            "actionId": "aws:fis:inject-api-throttle-error",
            "description": "Throttle APIs for 5 minutes",
            "parameters": {
                "service": "ec2",
                "operations": "DescribeInstances,DescribeVolumes",
                "percentage": "100",
                "duration": "PT2M"
            },
            "targets": {
                "Roles": "myRole"
            }
        }
    },
    "stopConditions": [
        {
            "source": "aws:cloudwatch:alarm",
            "value": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name"
}
```

Experiments for AWS FIS

AWS FIS enables you to perform fault injection experiments on your AWS workloads. To get started, create an experiment template (p. 22). After you create an experiment template, you can use it to start an experiment.

An experiment is finished when one of the following occurs:

- All actions (p. 22) in the template completed successfully.
- A stop condition (p. 28) is triggered.
- An action cannot be completed because of an error. For example, if the target (p. 23) cannot be found.
- The experiment is stopped manually (p. 39).

You cannot resume a stopped or failed experiment. You also cannot rerun a completed experiment. However, you can start a new experiment from the same experiment template. You can optionally update the experiment template before you specify it again in a new experiment.

Tasks

- Start an experiment (p. 37)
- View your experiments (p. 37)
- Tag an experiment (p. 38)
- Stop an experiment (p. 39)

Start an experiment

You start an experiment from an experiment template. For more information, see Start an experiment from a template (p. 31).

To monitor your experiment, you can do the following:

- View your experiments in the AWS FIS console. For more information, see View your experiments (p. 37).
- View Amazon CloudWatch metrics for the target resources in your experiments, or monitor AWS FIS usage metrics. For more information, see Monitor AWS FIS usage metrics with Amazon CloudWatch (p. 40).

View your experiments

You can view the progress of a running experiment, and you can view experiments that have completed, stopped, or failed.

Stopped, completed, and failed experiments are automatically removed from your account after 120 days.

To view experiments using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiments**.

3. To view information about a specific experiment, choose the **Experiment ID**.
4. To view the state of the experiment, check **State** in the **Details** pane. For more information, see experiment states (p. 38).

To view experiments using the CLI

Use the list-experiments command to get a list of experiments, and use the get-experiment command to get information about a specific experiment.

Experiment states

An experiment can be in one of the following states:

- **pending** – The experiment is pending.
- **initiating** – The experiment is preparing to start.
- **running** – The experiment is running.
- **completed** – All actions in the experiment completed successfully.
- **stopping** – The stop condition was triggered or the experiment was stopped manually.
- **stopped** – All running or pending actions in the experiment are stopped.
- **failed** – The experiment failed due to an error, such as insufficient permissions or incorrect syntax.

Action states

An action can be in one of the following states:

- **pending** – The action is pending, either because the experiment hasn't started or the action is to start later in the experiment.
- **initiating** – The action is preparing to start.
- **running** – The action is running.
- **completed** – The action completed successfully.
- **cancelled** – The experiment stopped before the action started.
- **stopping** – The action is stopping.
- **stopped** – All running or pending actions in the experiment are stopped.
- **failed** – The action failed due to a client error, such as insufficient permissions or incorrect syntax.

Tag an experiment

You can apply tags to experiments to help you organize them. You can also implement tag-based IAM policies (p. 52) to control access to experiments.

To tag an experiment using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiments**.
3. Select the experiment and choose **Actions**, **Manage tags**.
4. To add a new tag, choose **Add new tag**, and specify a key and value.

   To remove a tag, choose **Remove** for the tag.
5. Choose **Save**.

To tag an experiment using the CLI

Use the tag-resource command.

Stop an experiment

You can stop a running experiment at any time. When you stop an experiment, any post actions that have not been completed for an action are completed before the experiment stops. You cannot resume a stopped experiment.

To stop an experiment using the console

1. Open the AWS FIS console at https://console.aws.amazon.com/fis/.
2. In the navigation pane, choose **Experiments**.
3. Select the experiment, and choose **Stop experiment**.
4. In the confirmation dialog box, choose **Stop experiment**.

To stop an experiment using the CLI

Use the stop-experiment command.

Monitoring AWS FIS

You can use the following tools to monitor the progress and impact of your AWS Fault Injection Simulator (AWS FIS) experiments.

AWS FIS console and AWS CLI

You can use the AWS FIS console or the AWS CLI to monitor the progress of a running experiment. You can view the status of each action in the experiment, and the results of each action. For more information, see View your experiments (p. 37).

CloudWatch usage metrics and alarms

You can use CloudWatch usage metrics to provide visibility into your account's usage of resources. AWS FIS usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information, see Monitor AWS FIS usage metrics with Amazon CloudWatch (p. 40).

You can also create stop conditions for your AWS FIS experiments by creating CloudWatch alarms that define when an experiment goes out of bounds. When the alarm is triggered, the experiment is stopped. For more information, see Stop conditions for AWS FIS (p. 28). For more information about creating CloudWatch alarms, see Create a CloudWatch Alarm Based on a Static Threshold and Creating a CloudWatch Alarm Based on Anomaly Detection in the *Amazon CloudWatch User Guide*.

CloudTrail logs

You can use AWS CloudTrail to capture detailed information about the calls made to the AWS FIS API and store them as log files in Amazon S3. CloudTrail also logs calls made to service APIs for the resources on which you're running experiments. You can use these CloudTrail logs to determine which calls were made, the source IP address where the call came from, who made the call, when the call was made, and so on.

Monitor AWS FIS usage metrics with Amazon CloudWatch

You can use Amazon CloudWatch to monitor the impact of AWS FIS experiments on targets. You can also monitor your AWS FIS usage.

For more information about viewing the state of an experiment, see View your experiments (p. 37).

Monitor AWS FIS experiments

As you plan your AWS FIS experiments, identify the CloudWatch metrics that you can use to identify the baseline or "steady state" for the target resource types for the experiment. After you start an experiment, you can monitor those CloudWatch metrics for the targets selected through the experiment template.

For more information about the available CloudWatch metrics for a target resource type supported by AWS FIS, see the following:

- Monitor your instances using CloudWatch
- Amazon ECS CloudWatch metrics
- Overview of monitoring Amazon RDS

AWS FIS usage metrics

You can use CloudWatch usage metrics to provide visibility into your account's usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

AWS FIS usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about CloudWatch alarms, see the Amazon CloudWatch User Guide.

AWS FIS publishes the following metric in the **AWS/Usage** namespace.

Metric	Description
`ResourceCount`	The total number of the specified resource running on your account. The resource is defined by the dimensions associated with the metric.

The following dimensions are used to refine the usage metrics that are published by AWS FIS.

Dimension	Description
`Service`	The name of the AWS service containing the resource. For AWS FIS usage metrics, the value for this dimension is `FIS`.
`Type`	The type of entity that is being reported. Currently, the only valid value for AWS FIS usage metrics is `Resource`.
`Resource`	The type of resource that is running. The possible values are `ExperimentTemplates` for experiment templates, and `ActiveExperiments` for active experiments.
`Class`	This dimension is reserved for future use.

Log API calls with AWS CloudTrail

AWS Fault Injection Simulator (AWS FIS) is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, a role, or an AWS service in AWS FIS. CloudTrail captures all API calls for AWS FIS as events. The calls that are captured include calls from the AWS FIS console and code calls to the AWS FIS API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS FIS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS FIS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Use CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS FIS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**.

You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS FIS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Create a Trail for Your AWS Account
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All AWS FIS actions are logged by CloudTrail and are documented in the AWS Fault Injection Simulator API Reference. For the experiment actions that are carried out on a target resource, view the API reference documentation for the service that owns the resource. For example, for actions that are carried out on an Amazon EC2 instance, see the Amazon EC2 API Reference.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understand AWS FIS log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `StopExperiment` action.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAIOSFODNN7EXAMPLE:jdoe",
    "arn": "arn:aws:sts::111122223333:assumed-role/example/jdoe",
    "accountId": "111122223333",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/example",
        "accountId": "111122223333",
        "userName": "example"
      },
```

```
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-12-03T09:40:42Z"
      }
    }
  },
  "eventTime": "2020-12-03T09:44:20Z",
  "eventSource": "fis.amazonaws.com",
  "eventName": "StopExperiment",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "XXX.XXX.XXX.XXX",
  "userAgent": "userAgent",
  "requestParameters": {
    "id": "1234abc5-6def-789g-012h-ijklm34no56p"
  },
  "responseElements": {
    "experiment": {
      "actions": {
        "exampleAction1": {
          "actionId": "aws:ec2:stop-instances",
          "duration": "PT10M",
          "state": {
            "reason": "Initial state",
            "status": "PENDING"
          },
          "targets": {
            "Instances": "exampleTag1"
          }
        },
        "exampleAction2": {
          "actionId": "aws:ec2:stop-instances",
          "duration": "PT10M",
          "state": {
            "reason": "Initial state",
            "status": "PENDING"
          },
          "targets": {
            "Instances": "exampleTag2"
          }
        }
      },
      "creationTime": 1605788649.95,
      "endTime": 1606988660.846,
      "experimentTemplateId": "12345a67-8b9c-01d2-3e45-678f901gh2i3",
      "id": "1234abc5-6def-789g-012h-ijklm34no56p",
      "roleArn": "arn:aws:iam::111122223333:role/example",
      "startTime": 1605788650.109,
      "state": {
        "reason": "Example stopped",
        "status": "STOPPING"
      },
      "stopConditions": [
        {
          "source": "aws:cloudwatch:alarm",
          "value": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:example"
        }
      ],
      "tags": {
        "resourceArn": "arn:aws:fis:us-
east-1:111122223333:experiment/1234abc5-6def-789g-012h-ijklm34no56p"
      },
      "targets": {
        "ExampleTag1": {
          "resourceTags": {
            "Example": "tag1"
```

```
          },
          "resourceType": "aws:ec2:instance",
          "selectionMode": "RANDOM(1)"
        },
        "ExampleTag2": {
          "resourceTags": {
            "Example": "tag2"
          },
          "resourceType": "aws:ec2:instance",
          "selectionMode": "RANDOM(1)"
        }
      }
    }
  },
  "requestID": "1abcd23e-f4gh-567j-klm8-9np01q234r56",
  "eventID": "1234a56b-c78d-9e0f-g1h2-34jk56m7n890",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "111122223333"
}
```

Security in AWS Fault Injection Simulator

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Fault Injection Simulator, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS FIS. The following topics show you how to configure AWS FIS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS FIS resources.

Contents

Data protection in AWS Fault Injection Simulator

The AWS shared responsibility model applies to data protection in AWS Fault Injection Simulator. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.

- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with AWS FIS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

AWS FIS always encrypts your data at rest. Data in AWS FIS is encrypted at rest using transparent server-side encryption. This helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet encryption compliance and regulatory requirements.

Encryption in transit

AWS FIS encrypts data in transit between the service and other integrated AWS services. All data that passes between AWS FIS and integrated services is encrypted using Transport Layer Security (TLS). For more information about other integrated AWS services, see AWS FIS actions and supported AWS services (p. 2).

Identity and access management for AWS Fault Injection Simulator

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Fault Injection Simulator resources. IAM is an AWS service that you can use with no additional charge.

Contents

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Fault Injection Simulator.

Service user – If you use the AWS Fault Injection Simulator service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Fault Injection Simulator features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS Fault Injection Simulator, see Troubleshooting AWS Fault Injection Simulator identity and access (p. 57).

Service administrator – If you're in charge of AWS Fault Injection Simulator resources at your company, you probably have full access to AWS Fault Injection Simulator. It's your job to determine which AWS Fault Injection Simulator features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Fault Injection Simulator, see How AWS Fault Injection Simulator works with IAM (p. 50).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Fault Injection Simulator. To view example AWS Fault Injection Simulator identity-based policies that you can use in IAM, see AWS Fault Injection Simulator policy examples (p. 53).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for AWS Fault Injection Simulator in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear

in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Fault Injection Simulator works with IAM

Before you use IAM to manage access to AWS FIS, you should understand what IAM features are available to use with AWS FIS. To get a high-level view of how AWS FIS and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Contents

AWS FIS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. AWS FIS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in AWS FIS use the following prefix before the action: `fis:`. For example, to grant someone permission to create an experiment template, you include the `fis:CreateExperimentTemplate` action in their policy. Policy statements must include either an `Action` or `NotAction` element. AWS FIS defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "fis:action1",
      "fis:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "fis:List*"
```

To see a list of AWS FIS actions, see Actions Defined by AWS Fault Injection Simulator in the *Service Authorization Reference*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

For example, an experiment template has the following ARN:

```
arn:${Partition}:fis:${Region}:${Account}:experiment-template/${experimentTemplateId}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

Some AWS FIS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Some AWS FIS API actions involve multiple resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of AWS FIS resource types and their ARNs, see Resources Defined by AWS Fault Injection Simulator in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS Fault Injection Simulator.

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS FIS condition keys, see Condition Keys for AWS Fault Injection Simulator in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Fault Injection Simulator.

Resource-based policies

AWS FIS does not support resource-based policies.

Authorization based on AWS FIS tags

You can attach tags to AWS FIS resources or pass tags in a request to AWS FIS. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys.

For an example of a policy, see Example: Use tags to control the use of resources (p. 56).

AWS FIS IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with AWS FIS

You can use temporary credentials to sign in with federation, to assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations, such as AssumeRole or GetFederationToken.

AWS FIS supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

AWS FIS supports service-linked roles. For details about creating or managing AWS FIS service-linked roles, see Use service-linked roles for AWS Fault Injection Simulator (p. 59).

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

AWS FIS supports service roles.

AWS Fault Injection Simulator policy examples

By default, IAM users and roles don't have permission to create or modify AWS FIS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Examples

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS Fault Injection Simulator resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using AWS Fault Injection Simulator quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Example: Use the AWS FIS console

To access the AWS Fault Injection Simulator console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS FIS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

The following policy grants users permission to list and view all resources in the AWS FIS console, but not to create, update, or delete them.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "fis:List*",
                "fis:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Example: List available AWS FIS actions

The following policy grants users permission to list the available AWS FIS actions.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "fis:ListActions"
            ],
            "Resource": "arn:aws:fis:*:*:action/*"
        }
    ]
}
```

Example: Create an experiment template for a specific action

The following policy grants users permission to create an experiment template for the action `aws:ec2:stop-instances`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyExample",
      "Effect": "Allow",
      "Action": [
          "fis:CreateExperimentTemplate"
      ],
      "Resource": [
        "arn:aws:fis:*:*:action/aws:ec2:stop-instances",
        "arn:aws:fis:*:*:experiment-template/*"
      ]
    },
    {
      "Sid": "PolicyPassRoleExample",
      "Effect": "Allow",
      "Action": [
          "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::111122223333:role/roleName"
      ]
    }
  ]
}
```

Example: Start an experiment

The following policy grants users permission to start an experiment using the specified IAM role and experiment template. It also allows AWS FIS to create a service-linked role on the user's behalf. For more information, see Use service-linked roles for AWS Fault Injection Simulator (p. 59).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyExample",
      "Effect": "Allow",
      "Action": [
          "fis:StartExperiment"
      ],
      "Resource": [
        "arn:aws:fis:*:*:experiment-template/experimentTemplateID",
```

```
            "arn:aws:fis:*:*:experiment/*"
        ]
    },
    {
        "Sid": "PolicyExampleforServiceLinkedRole",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "fis.amazonaws.com"
            }
        }
    }
  ]
}
```

Example: Use tags to control the use of resources

You might want to control which AWS FIS resources users can use. For example, the following policy allows users to run experiments from experiment templates that have the tag `Purpose=Test`. Users cannot create or modify experiment templates, and they cannot run experiments using templates that do not have the specified tag.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "fis:StartExperiment",
            "Resource": "arn:aws:fis:*:*:experiment-template/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Purpose": "Test"
                }
            }
        }
    ]
}
```

Example: Delete an experiment template with a specific tag

The following policy grants users permission to delete an experiment template with tag `Purpose=Test`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "fis:DeleteExperimentTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Purpose": "Test"
                }
            }
        }
    ]
}
```

Example: Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting AWS Fault Injection Simulator identity and access

Use the following information to help you diagnose and fix common access issues that you might encounter when working with AWS FIS and IAM.

Issues

I am not authorized to perform an action in AWS FIS

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view experiments, but does not have `fis:ListExperiments` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 fis:ListExperiments on resource: my-experiment
```

In this case, Mateo asks their administrator to update their policies to allow them to access the *my-experiment* resource using the `fis:ListExperiments` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS Fault Injection Simulator.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS Fault Injection Simulator. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access AWS FIS

To allow others to access AWS Fault Injection Simulator, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS Fault Injection Simulator.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my AWS FIS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS Fault Injection Simulator supports these features, see How AWS Fault Injection Simulator works with IAM (p. 50).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Use service-linked roles for AWS Fault Injection Simulator

AWS Fault Injection Simulator uses AWS Identity and Access Management (IAM)service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS FIS. Service-linked roles are predefined by AWS FIS and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS FIS easier because you don't have to manually add the necessary permissions to manage monitoring and resource selection for experiments. AWS FIS defines the permissions of its service-linked roles, and unless defined otherwise, only AWS FIS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

> **Note**
> In addition to the service-linked role, you must also specify an IAM role that grants permission to modify the resources that you specify as targets in an experiment template. For more information, see Step 2: Set up the IAM role for the AWS FIS service (p. 7).

You can delete a service-linked role only after first deleting the related resources. This protects your AWS FIS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM, and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS FIS

AWS FIS uses the service-linked role named **AWSServiceRoleForFIS** to enable it to manage monitoring and resource selection for experiments.

The **AWSServiceRoleForFIS** service-linked role trusts the following services to assume the role:

- `fis.amazonaws.com`

The **AWSServiceRoleForFIS** service-linked role uses the managed policy AmazonFISServiceRolePolicy (p. 61). This policy allows AWS FIS to complete the following actions on the following resources:

- Actions `events:PutRule`, `events:DeleteRule`, `events:DescribeRule`, `events:PutTargets`, and `events:RemoveTargets` on all resources that match the following condition:

```
{ "StringEquals" : { "events:ManagedBy" : "fis.amazonaws.com" }}
```

- Action `tag:GetResources` on all resources
- Actions `cloudwatch:DescribeAlarms` and `cloudwatch:DescribeAlarmHistory` on all resources
- Actions `iam:GetUser`, `iam:GetRole`, `iam:ListUsers`, and `iam:ListRoles` on all resources
- Action `ec2:DescribeInstances` on all resources
- Action `ecs:DescribeClusters` on all resources
- Action `eks:DescribeNodegroup` on all resources
- Actions `rds:DescribeDBClusters` and `rds:DescribeDBInstances` on all resources

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For the **AWSServiceRoleForFIS** service-linked role to be successfully created, the IAM identity that you use AWS FIS with must have the required permissions. To grant the required permissions, attach the following policy to the IAM identity.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "fis.amazonaws.com"
                }
            }
        }
    ]
}
```

For more information, see Service-linked role permissions in the *IAM User Guide*.

Create a service-linked role for AWS FIS

You don't need to manually create a service-linked role. When you start an AWS FIS experiment in the AWS Management Console, the AWS CLI, or the AWS API, AWS FIS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you start an AWS FIS experiment, AWS FIS creates the service-linked role for you again.

Edit a service-linked role for AWS FIS

AWS FIS does not allow you to edit the **AWSServiceRoleForFIS** service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the

role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Delete a service-linked role for AWS FIS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
> If the AWS FIS service is using the role when you try to clean up the resources, then the cleanup might fail. If that happens, wait for a few minutes and try the operation again.

To clean up AWS FIS resources used by the AWSServiceRoleForFIS

Make sure that none of your experiments are currently running. If necessary, stop your experiments. For more information, see Stop an experiment (p. 39).

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForFIS** service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for AWS FIS service-linked roles

AWS FIS supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Fault Injection Simulator endpoints and quotas.

AWS managed policies for AWS Fault Injection Simulator

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AmazonFISServiceRolePolicy

This policy is attached to the service-linked role named **AWSServiceRoleForFIS** to allow AWS FIS to manage monitoring and resource selection for experiments. For more information, see Use service-linked roles for AWS Fault Injection Simulator (p. 59).

AWS FIS updates to AWS managed policies

View details about updates to AWS managed policies for AWS FIS since this service began tracking these changes.

Change	Description	Date
AmazonFISServiceRolePolicy (p. 61) - Update to an existing policy	Added an action to allow AWS FIS to retrieve history for the CloudWatch alarms used in stop conditions.	June 30, 2021
AWS FIS started tracking changes	AWS FIS started tracking changes to its AWS managed policies	March 1, 2021

Infrastructure security in AWS Fault Injection Simulator

As a managed service, AWS Fault Injection Simulator (AWS FIS) is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS FIS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

AWS Fault Injection Simulator and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and AWS Fault Injection Simulator by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access AWS FIS APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS FIS APIs. Traffic between your VPC and AWS FIS does not leave the Amazon network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *AWS PrivateLink Guide*.

Considerations for AWS FIS VPC endpoints

Before you set up an interface VPC endpoint for AWS FIS, ensure that you review Interface endpoint properties and limitations in the *AWS PrivateLink Guide*.

AWS FIS supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for AWS FIS

You can create a VPC endpoint for the AWS FIS service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *AWS PrivateLink Guide*.

Create a VPC endpoint for AWS FIS using the following service name: `com.amazonaws.region.fis`.

If you enable private DNS for the endpoint, you can make API requests to AWS FIS using its default DNS name for the Region, for example, `fis.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *AWS PrivateLink Guide*.

Creating a VPC endpoint policy for AWS FIS

You can attach an endpoint policy to your VPC endpoint that controls access to AWS FIS. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for specific AWS FIS actions

The following VPC endpoint policy grants access to the listed AWS FIS actions on all resources to all principals.

```
{
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "fis:ListExperimentTemplates",
            "fis:StartExperiment",
            "fis:StopExperiment",
            "fis:GetExperiment"
         ],
         "Resource":"*",
         "Principal":"*"
      }
   ]
}
```

Example: VPC endpoint policy that denies access from a specific AWS account

The following VPC endpoint policy denies the specified AWS account access to all actions and resources, but grants all other AWS accounts access to all actions and resources.

```
{
   "Statement":[
      {
```

```
         "Effect": "Allow",
         "Action": "*",
         "Resource": "*",
         "Principal": "*"
      },
      {
         "Effect":"Deny",
         "Action": "*",
         "Resource": "*",
         "Principal": {
           "AWS": [ "123456789012" ]
         }
      }
   ]
}
```

Tag your AWS FIS resources

A *tag* is a metadata label that either you or AWS assigns to an AWS resource. Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and the value. For example, you might define the key as `purpose` and the value as `test` for a resource.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.
- Control access to your AWS resources. For more information, see Controlling Access Using Tags in the *IAM User Guide*.

Supported resources

The following AWS Fault Injection Simulator (AWS FIS) resources support tagging:

- Actions
- Experiments
- Experiment templates

Tagging restrictions

The following basic restrictions apply to tags on AWS FIS resources:

- Maximum number of tags that you can assign to a resource: 50
- Maximum key length: 128 Unicode characters
- Maximum value length: 256 Unicode characters
- Valid characters for keys and values: a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - and @
- Keys and values are case sensitive
- You cannot use `aws:` as a prefix for keys, because it's reserved for AWS use

Work with tags

You can work with tags using the AWS FIS console or the command line.

For more information about tagging your AWS FIS resources using the console, see the following topics:

- Tag an experiment (p. 38)
- Tag experiment templates (p. 31)

You can use the following AWS CLI commands to work with tags:

- Use the tag-resource command to tag a resource.
- Use the untag-resource command to remove tags from a resource.

- Use the list-tags-for-resource command to list the tags for a specific resource.

Quotas for AWS Fault Injection Simulator

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, but not for all quotas.

To view the quotas for AWS FIS, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **AWS Fault Injection Simulator**.

To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

Your AWS account has the following quotas related to AWS FIS.

Name	Default	Adjustable
Action duration	12 hours	No
Actions per experiment template	20	No
Active experiments	5	No
Completed experiment data retention	120 days	No
Experiment duration	12 hours	No
Experiment templates	12	No
Parallel actions per experiment	10	No
Resources per experiment target	5	No
Stop conditions per experiment template	5	No

Document history

The following table describes important documentation updates in the *AWS Fault Injection Simulator User Guide*.

update-history-change	update-history-description	update-history-date
Initial release (p. 68)	The initial release of the AWS Fault Injection Simulator User Guide.	March 15, 2021