Getting started with AWS Fault Injection Simulator

Rohini Gaonkar

Sr. Developer Advocate, AWS

@rohini_gaonkar

@gaonkarr

Chaos Engineering

Experimenting on a software system to build confidence

Fundamental goals with chaos engineering

- Improve resilience and performance
- Uncover hidden issues
- Expose blind spots
  Monitoring, observability, and alarm
- And more

Why is chaos engineering difficult?

1 Stitch together different tools and homemade scripts

2 Agents or libraries required to get started

3 Difficult to ensure safety

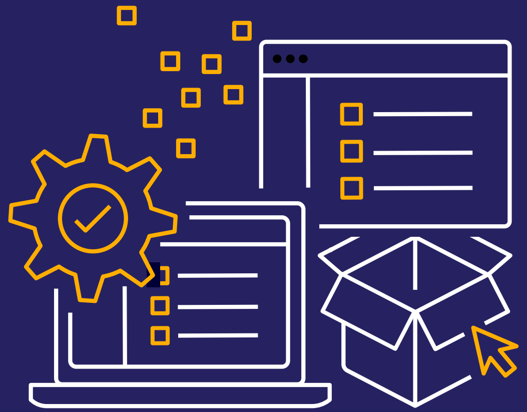
4 Difficult to reproduce “real-world” events (multiple failures at once)

Fully managed chaos engineering service

Easy to get started

Real-world conditions

Safeguards



Easy to get started

- No need to integrate multiple tools and homemade scripts or install agents
- Use the AWS Management Console or the AWS CLI
- Use pre-existing experiment templates and get started in minutes
- Easily share it with others

Real-world conditions

Run experiments in sequence of events or in parallel

Target all levels of the system (host, infrastructure, network, etc.)

Real faults injected at the service control plane level!

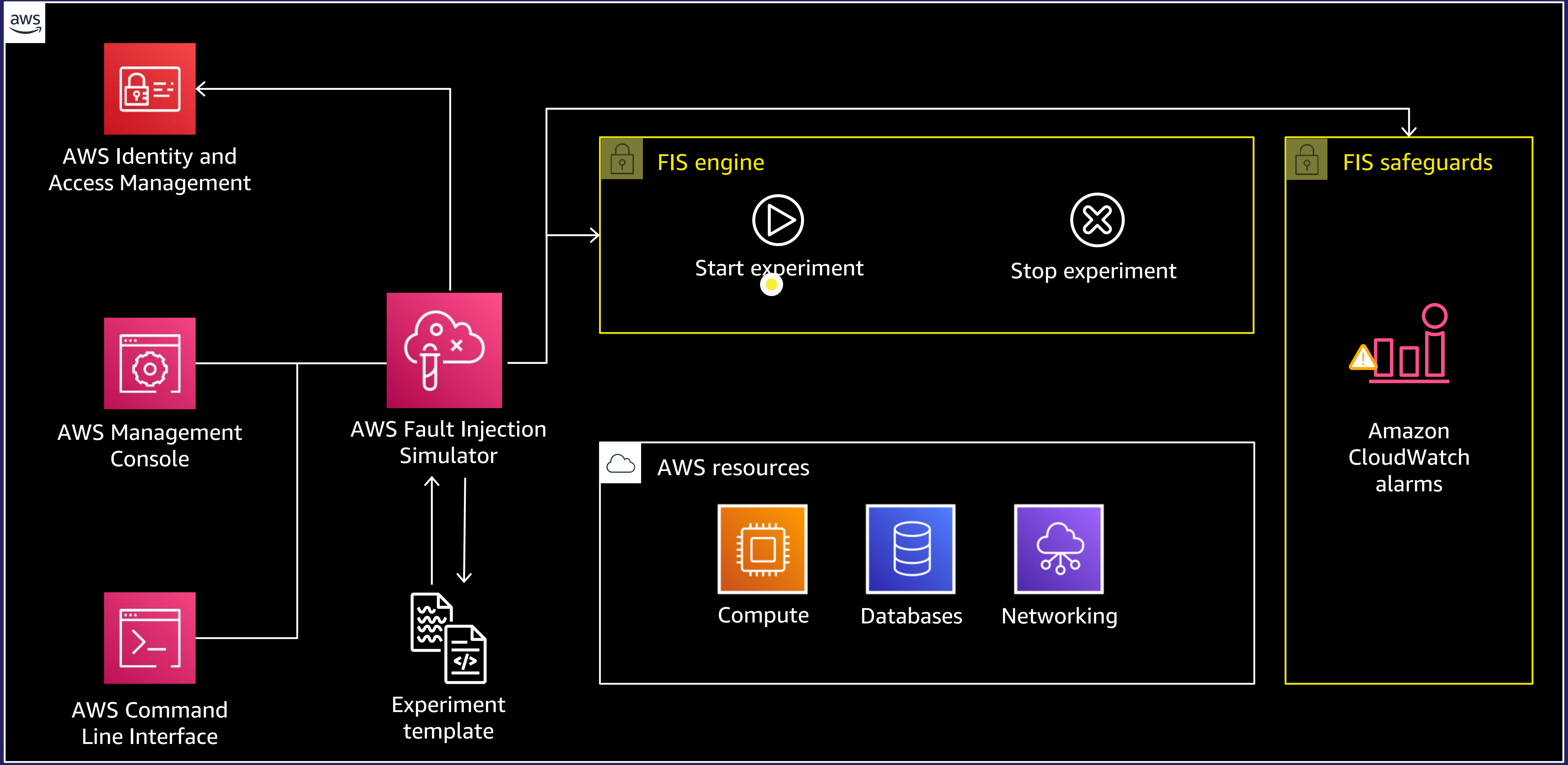
Safeguards

- “Stop conditions” alarms
- Integration with Amazon CloudWatch
- Built-in rollbacks
- Fine-grain IAM controls

Demo
Stop instance with tags

https://github.com/gaonkarr/aws-fis-demo

Architecture



Components

Actions

1

Targets

2

Experiment templates

3

Experiments

4

1

Actions

Actions are the fault injection actions executed during an experiment on a target resource

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

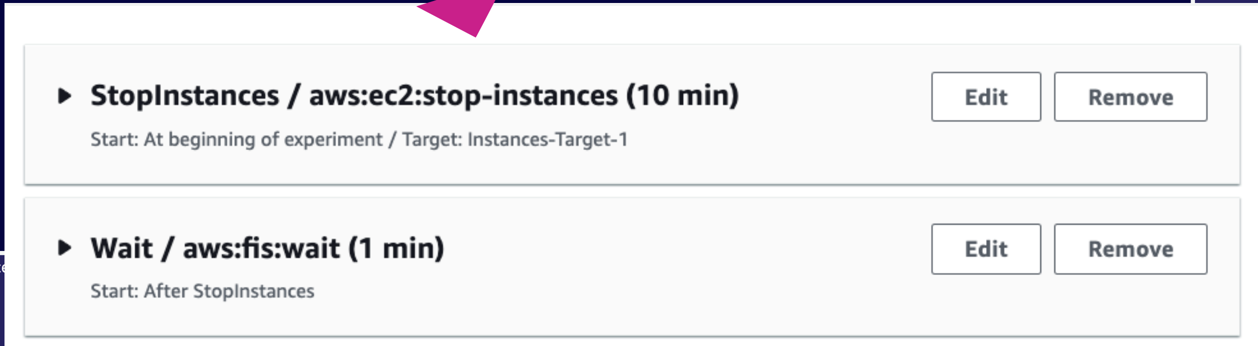
Actions include:

- Fault type
- Duration
- Targeted resources
- Timing relative to any other actions
- Fault-specific parameters, such as rollback behavior or the portion of requests to throttle

1

Actions

```
"actions": {
    "StopInstances": {
        "actionId": "aws:ec2:stop-instances",
        "parameters": {
            "duration": "PT10M",
            "startInstancesAtEnd": "true"
        },
        "targets": {
            "Instances": "RandomInstancesInAZ"
        }
    },
    ”Wait": {
        "actionId": " aws:fis:wait",
        "parameters": {
            "duration": "PT1M",
        },
        "startAfter": [
            "StopInstances"
        ]
    },
}
```



▸ **StopInstances / aws:ec2:stop-instances (10 min)** Edit Remove

Start: At beginning of experiment / Target: Instances-Target-1

▸ **Wait / aws:fis:wait (1 min)** Edit Remove

Start: After StopInstances

2

Targets

Targets define one or more AWS resources on which to carry out an action

Targets include:

- Resource type
- Resource IDs, tags, and filters
- Selection mode (e.g., ALL, COUNT, PERCENT)

2

Targets

```
    "targets": {
        ”RandomInstancesInAZ": {
            "resourceType": "aws:ec2:instance",
            "resourceTags": {
                “Purpose": “ChaosReady"
            },
            "filters" : [
                {
                    "path": "Placement.AvailabilityZone",
                    "values": ["us.east.1a"]
                },
                {
                    "path": "State.Name",
                    "values": ["running"]
                },
                {
                    "path": "VpcId",
                    "values": ["vpc-0123456789"]
                }
            ]
            "selectionMode": ”COUNT(2)"
        }
```

3

Experiment templates

Experiment templates define an experiment and are used in the start-experiment request

Experiment templates include:

- Actions
- Targets
- Stop condition alarms
- IAM role
- Description
- Tags

3

Experiment template

Name

Description

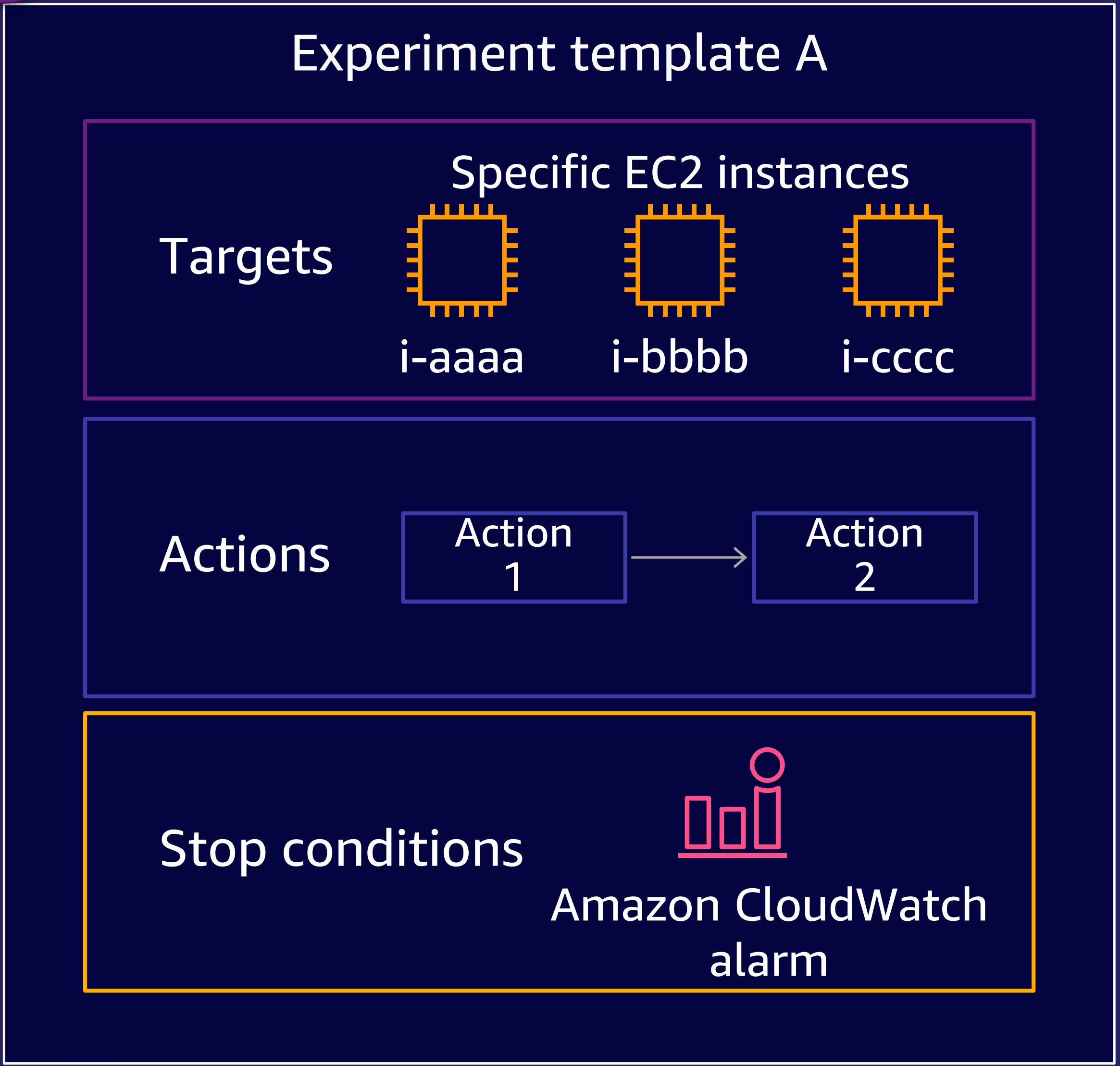
IAM role

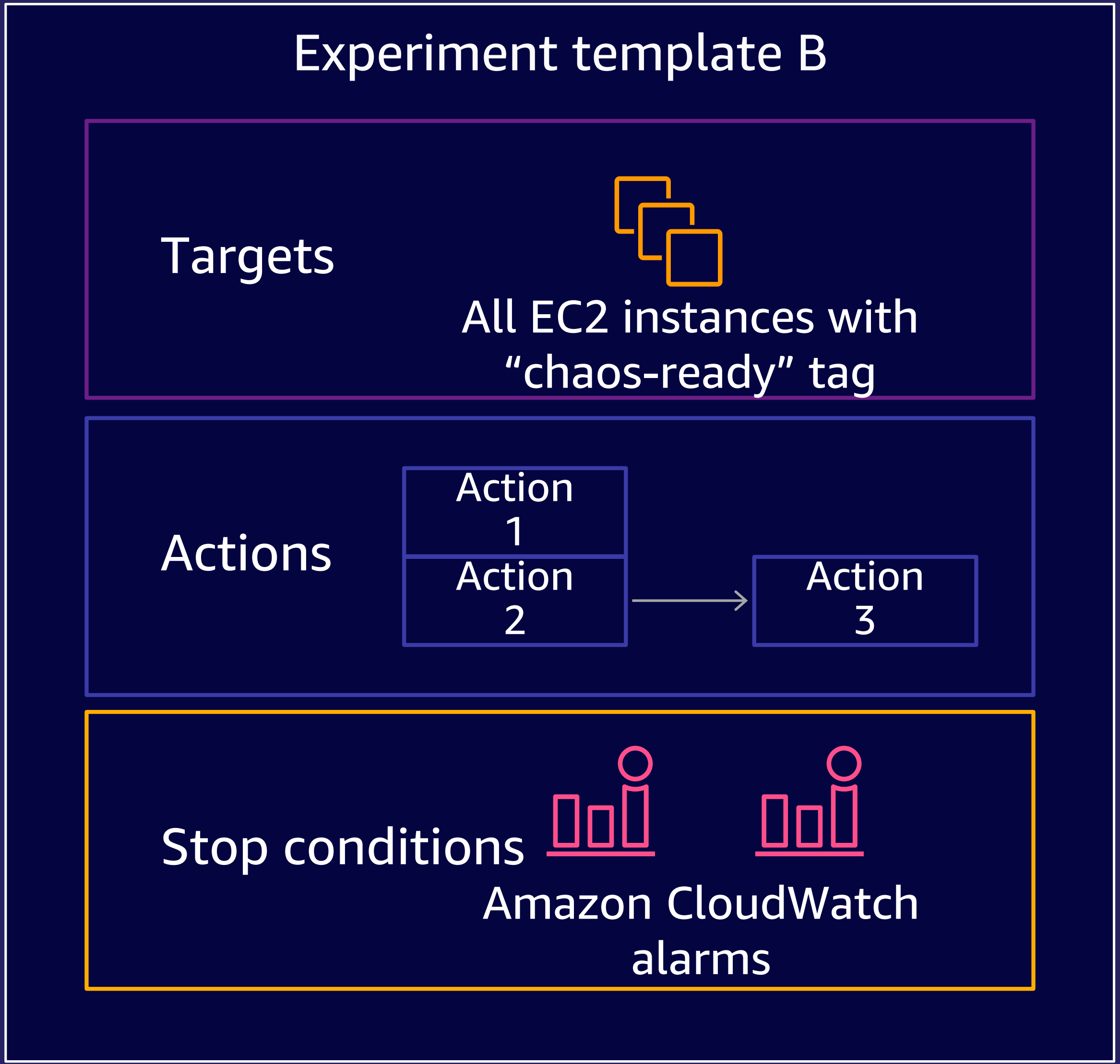
Stop conditions

Targets

Actions

```
{
  "tags": {  "Name": "StopAndRestartRandomeInstance”  },
   "description": "Stop and Restart One Random Instance",
   "roleArn": "arn:aws:iam::0123456789:role/MyFISExperimentRole”,
   "stopConditions": [
       {
            "source": "aws:cloudwatch:alarm",
            "value": " "arn:aws:cloudwatch:us-east 1:0123456789:alarm:No_Traffic"
       }
  ],
   "targets": {
        "myInstance": {
            "resourceTags": {   ”Env": ”test”    },
            "resourceType": "aws:ec2:instance",
            "selectionMode": ”COUNT(1)"
       }
  },
  "actions": {
        "StopInstances": {
            "actionId": "aws:ec2:stop-instances",
            "description": "stop the instances",
            "parameters": {
                "startInstancesAtEnd": "true”,
                "duration": "PT2M",
            },
            "targets": {
                "Instances": "myInstance"
        }
      }
    }
}
```


Experiment template A
Targets
Specific EC2 instances
i-aaaa
i-bbbb
i-cccc
Actions
Action 1
Action 2
Stop conditions
Amazon CloudWatch alarm


Experiment template B
Targets
All EC2 instances with "chaos-ready" tag
Actions
Action 1
Action 2
Action 3
Stop conditions
Amazon CloudWatch alarms

4

Experiments

Experiments are snapshot of the experiment template when it was first launched

Experiments include:

- Snapshot of the experiment
- Creation and start time
- Status
- Execution ID
- Experiment template ID
- IAM role ARN

Supported fault injections

- ✓ API throttling
- ✓ Server error (Amazon Elastic Compute Cloud (EC2))
- ✓ Stop, reboot, and terminate instance(s) (EC2)
- ✓ Increased memory or CPU load (EC2)
- ✓ Kill process (EC2)
- ✓ Latency injection (EC2)
- ✓ Container instance termination (Amazon Elastic Container Service (ECS))
- ✓ Increase memory or CPU consumption per task (ECS)
- ✓ Terminate nodes (Amazon Elastic Kubernetes Service (EKS))
- ✓ Database stop, reboot, and failover (Amazon Relational Database Service (RDS))
- ✓ And more to come…

Demos

Demo

Stop instance with tags and alarms

https://github.com/gaonkarr/aws-fis-demo

Demo
Stop instances with tags and filters

https://github.com/gaonkarr/aws-fis-demo

Resource filters

```
aws ec2 describe-instances
```

```
{
"Reservations": [ {"Groups": [],
        "Instances": [
          {
                "ImageId": "ami-0011111111111111",
                "InstanceId": "i-00aaaaaaaaaaaaaa",
                ...
                "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        ...
                },
                ...
                "PublicIpAddress": "203.0.113.17",
                "State": {
                        "Name": "running"
                        ...
                },
                "SubnetId": "subnet-abc123456",
                "VpcId": "vpc-00bbbbbb",
...
...
```

Resource filters

```
aws ec2 describe-instances
```

```
{
"Reservations": [ {"Groups": [],
        "Instances": [
          {
                "ImageId": "ami-0011111111111111",
                "InstanceId": "i-00aaaaaaaaaaaaaa",
                ...
                "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        ...
                },
                ...
                "PublicIpAddress": "203.0.113.17",
                "State": {
                        "Name": "running”
                        ...
                },
                "SubnetId": "subnet-abc123456",
                "VpcId": "vpc-00bbbbbb",
...
...
```

```
"filters": [
        {
        "path": ”ImageId",
        "values": [ ”ami-0011111111111111" ]
        }
],
```

Resource filters

```
aws ec2 describe-instances
```

```
{
"Reservations": [ {"Groups": [],
        "Instances": [
          {
                "ImageId": "ami-0011111111111111",
                "InstanceId": "i-00aaaaaaaaaaaaaa",
                ...
                "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        ...
                },
                ...
                "PublicIpAddress": "203.0.113.17",
                "State": {
                        "Name": "running"
                        ...
                },
                "SubnetId": "subnet-abc123456",
                "VpcId": "vpc-00bbbbbb",
...
...
```

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

Demo
CPU stress fault injection using AWS Systems Manager with alarms to stop-experiment via CLI

https://github.com/gaonkarr/aws-fis-demo

Resources

- AWS Fault Injection Simulator

https://aws.amazon.com/fis/

- AWS Blog AWS Fault Injection Simulator – Use Controlled Experiments to Boost Resilience

https://aws.amazon.com/blogs/aws/aws-fault-injection-simulator-use-controlled-experiments-to-boost-resilience/

- Demos

https://github.com/gaonkarr/aws-fis-demo

Thank you!

Rohini Gaonkar

Sr. Developer Advocate, AWS

@rohini_gaonkar

@gaonkarr