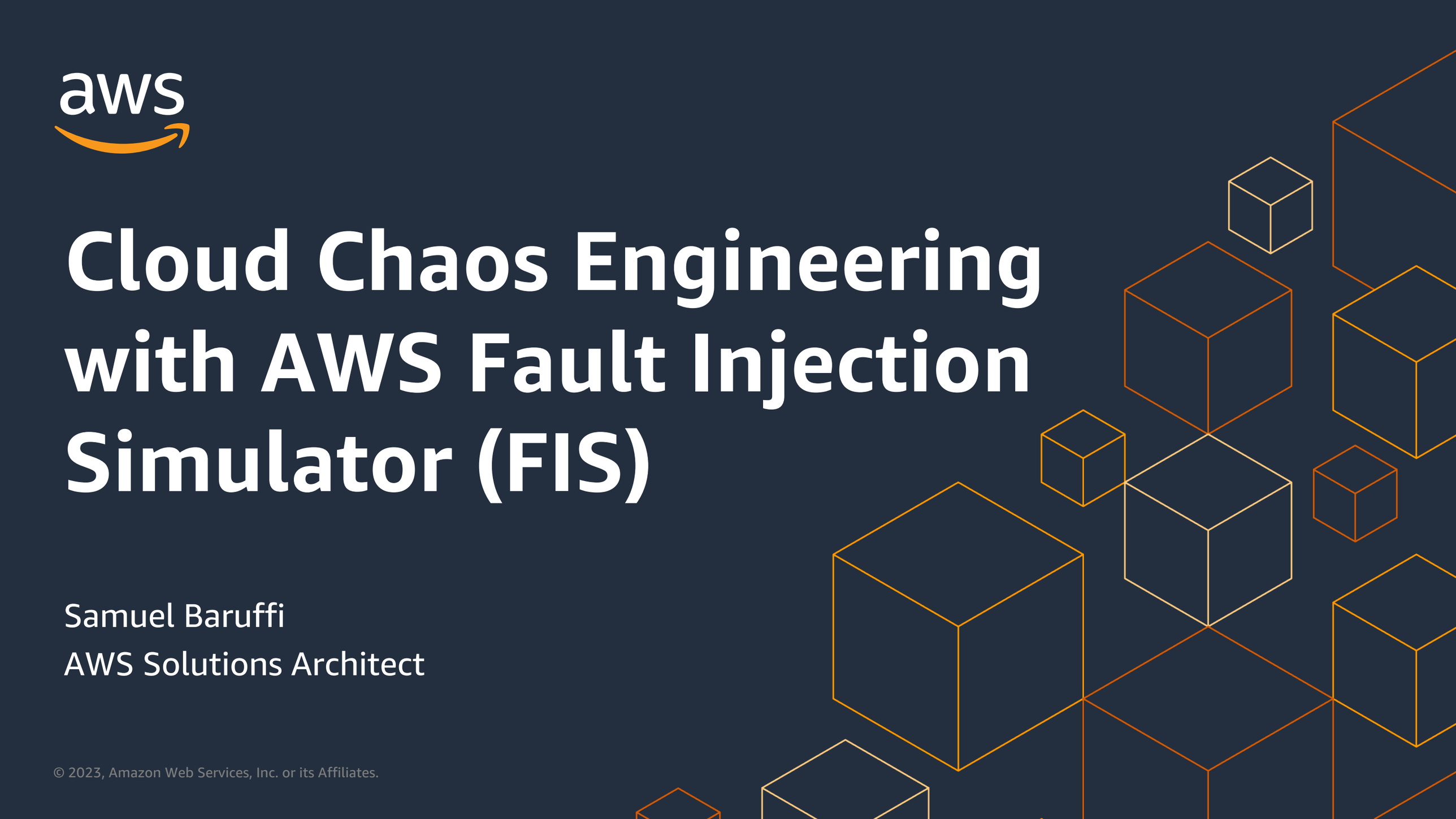


Cloud Chaos Engineering with AWS Fault Injection Simulator (FIS)

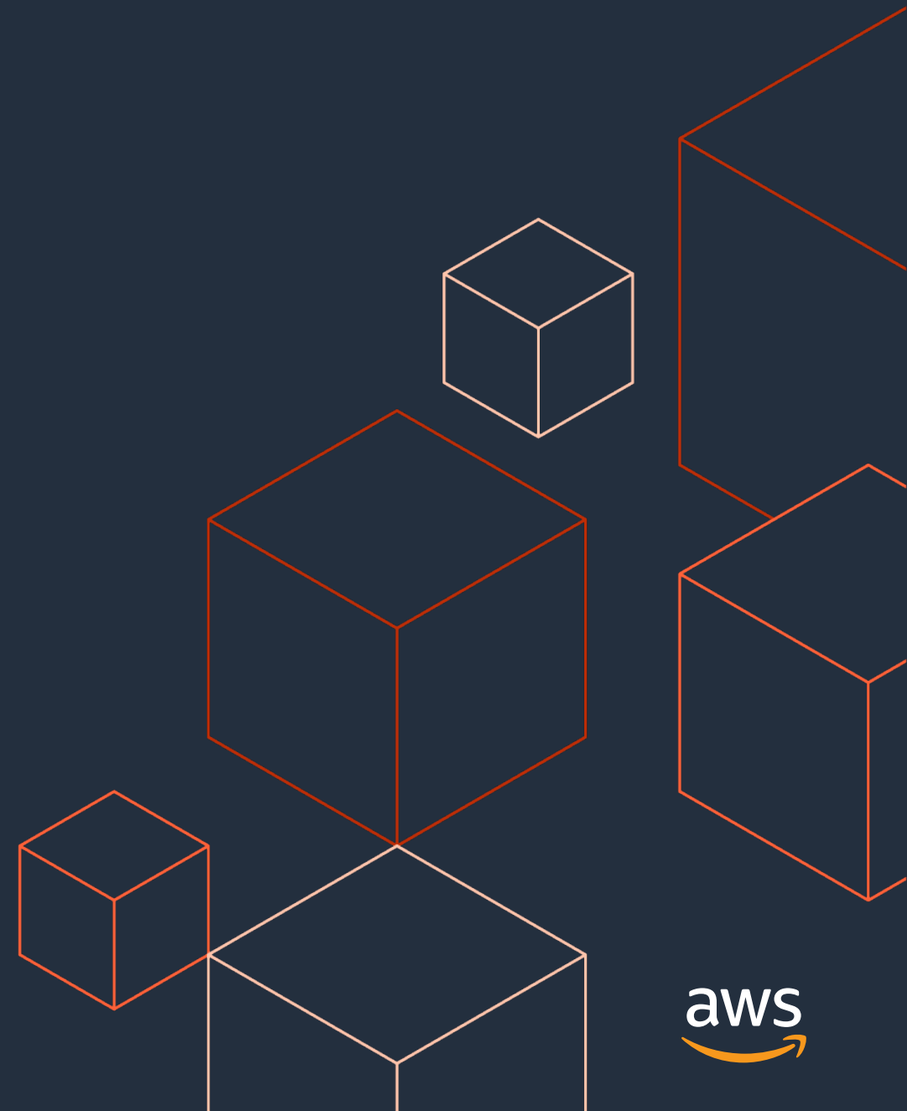
Samuel Baruffi
AWS Solutions Architect

Agenda

- Challenges with distributed systems
- Why is chaos engineering hard?
- Introducing AWS Fault Injection Simulator (FIS)
- Key features
- Use cases
- Demo

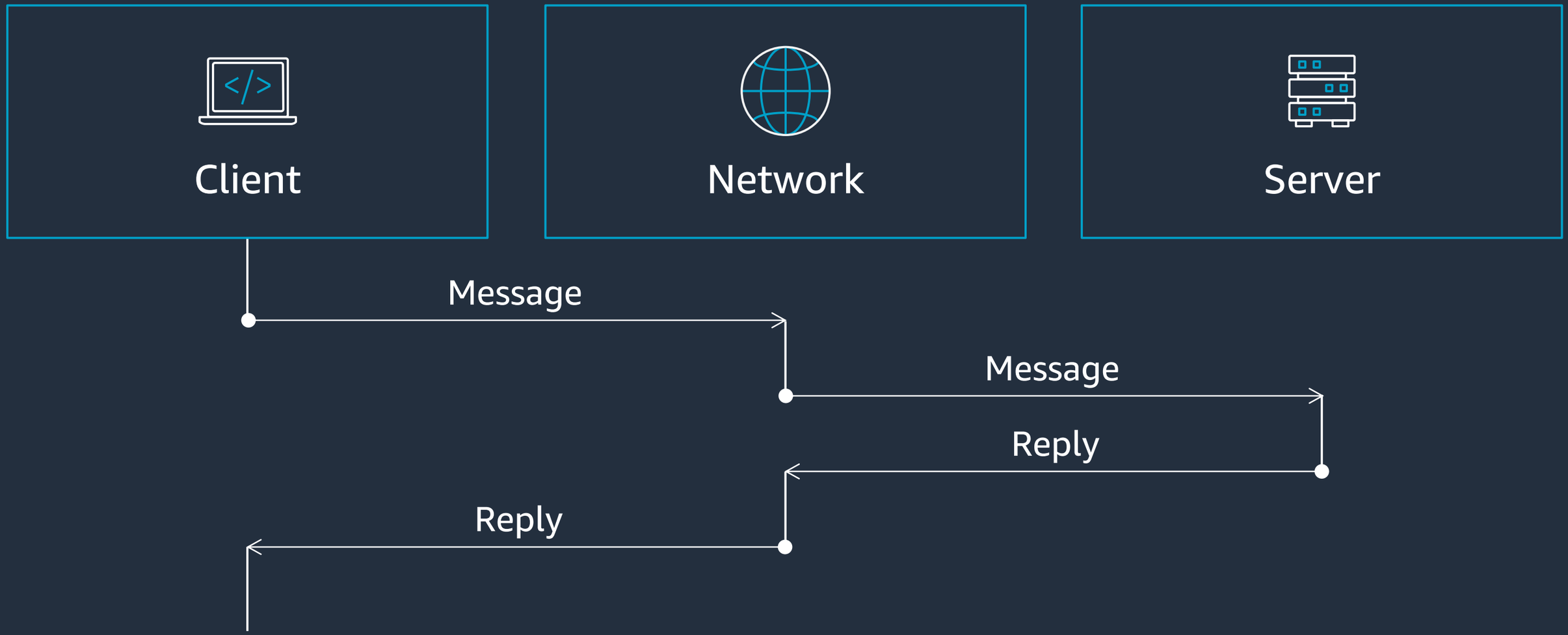
aws

Challenges with distributed systems



aws

Distributed systems are complex



https://aws.amazon.com/builders-library/challenges-with-distributed-systems/

Traditional testing is not enough

Unit testing
of components

Tested in isolation to ensure function meets expectations

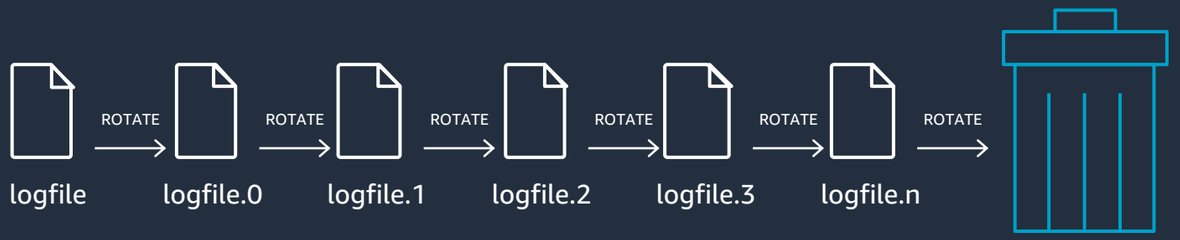
Functional testing
of integrations

Each execution path tested to assure expected results

TESTING = VERIFYING A KNOWN CONDITION

aws

And it can get more complicated…



```
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
close failed in file object destructor:
IOError: No space left on device
```

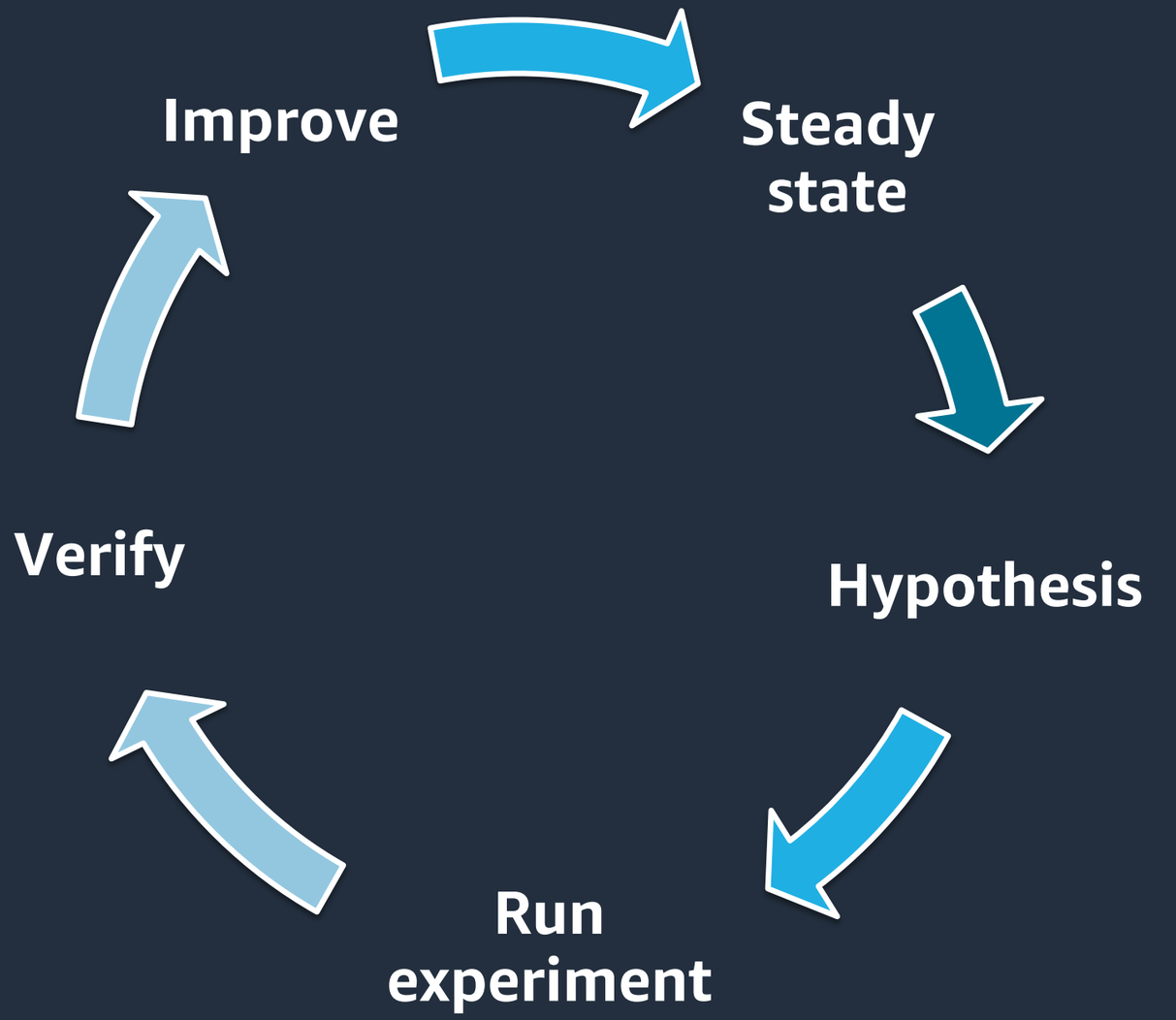

aws

Chaos engineering

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

aws

Phases of chaos engineering



aws

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

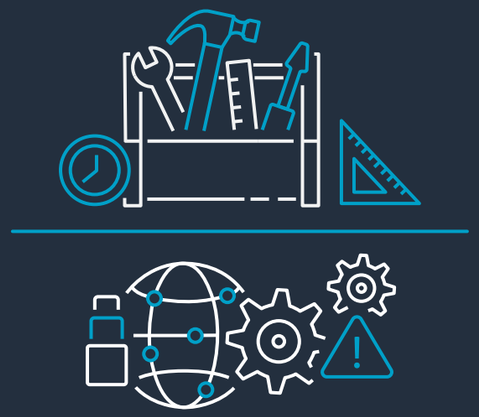
Difficult to reproduce “real-world” events (multiple failures at once)

aws

AWS Fault Injection Simulator

Fully managed chaos engineering service

Easy to
get started



Real-world
conditions

Safeguards

aws

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

aws

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!

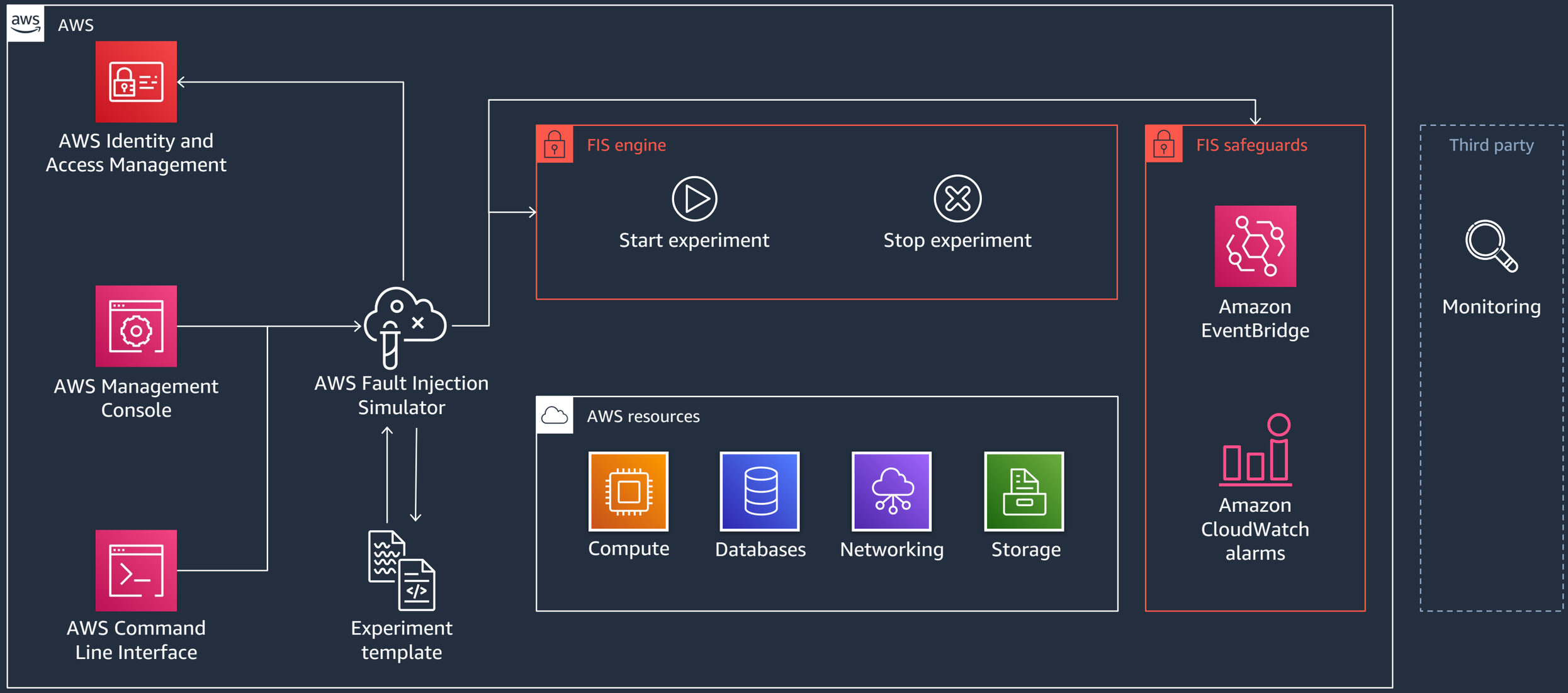
aws

Safeguards

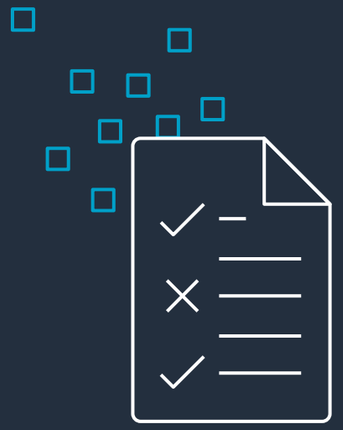
- “Stop conditions” alarms
- Integration with Amazon CloudWatch
- Built-in rollbacks
- Fine-grain IAM controls

aws

AWS Fault Injection Simulator



Components



Actions

Targets

Experiment templates

Experiments

aws

Actions

Actions are the fault injection actions executed during an experiment

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

Actions include:

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

aws

Actions

```
"actions": {
        "StopInstances": {
                "actionId": "aws:ec2:stop-instances",
                "parameters": {
                    "startInstancesAfterDuration": "PT2M"
                },
                "targets": {
                        "Instances": "RandomInstancesInAZ"
                }
        },
        ”Wait": {
                "actionId": " aws:fis:wait",
                "parameters": {
                    "duration": "PT1M",
                },
                "startAfter": [
                        "StopInstances"
                ]
        },

}
```


aws

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, RANDOM)

aws

Targets

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


aws

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

aws

Experiment templates

Name

Description

IAM role

Stop conditions

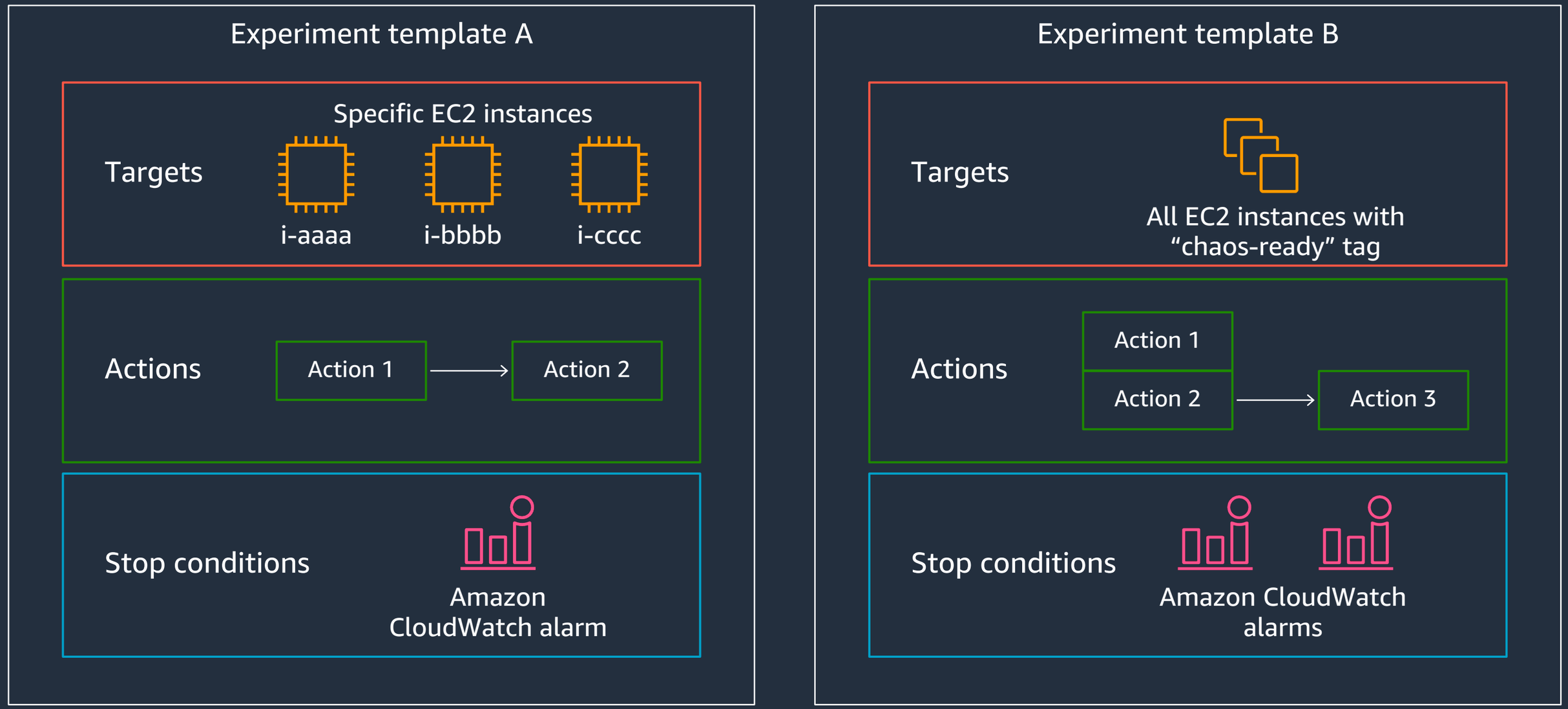
Targets

Actions

```
{
   "tags”: {
        "Name": "StopAndRestartRandomInstance"
   },
   "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"
            }
        }
    }
}
```


aws

Experiment templates



Experiments

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

Experiments include:

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

aws

Supported fault injections

- Server error (EC2)
- Stop, reboot, and terminate instance(s) (EC2)
- API throttling
- Increased memory or CPU load (EC2)
- Kill process (EC2)
- Latency injection (EC2)
- Container instance termination (ECS)
- Increase memory or CPU consumption per task (ECS)
- Terminate nodes (EKS)
- Database stop, reboot, and failover (RDS)
- And more (network disruption, EBS pause, and others)

aws

Use cases

Use cases

One-off experiments

Periodic game days

Automated experiments

aws

Use cases

One-off experiments

Periodic game days

Automated experiments

aws

Use cases

One-off experiments

Periodic game days

Automated experiments

aws

Use cases

One-off experiments

Periodic game days

Automated experiments

aws

Automated experiments

Recurring scheduled experiments

Event-triggered experiments

Continuous delivery experiments

aws

Automated experiments

Recurring scheduled experiments

Event-triggered experiments

Continuous delivery experiments

aws

Automated experiments

Recurring scheduled experiments

Event-triggered experiments

Continuous delivery experiments

aws

Automated experiments

Recurring scheduled experiments

Event-triggered experiments

Continuous delivery experiments

aws

Automated experiments

Recurring scheduled experiments

Event-triggered experiments

Continuous delivery experiments

aws

Use cases

One-off experiments

Periodic game days

Automated experiments

aws

Resources

AWS Well-Architected Framework
https://aws.amazon.com/architecture/well-architected/

AWS Fault Injection Simulator
https://aws.amazon.com/fis/

AWS Chaos Engineering Workshop
https://chaos-engineering.workshop.aws/

AWS FIS Documentation
https://docs.aws.amazon.com/fis/

AWS FIS Samples
https://github.com/aws-samples/aws-fault-injection-simulator-samples

aws

Demo

Demo

Stop instance

Thank you!

Samuel Baruffi
@samuelbaruffi