**UNIVERSIDAD DE LOS ANDES**

**MISO-4208 Pruebas Automáticas**

**Profesor**
**Mario Linares Vásquez - m.linaresv@uniandes.edu.co**

# TALLER 8 Netflix Simian Army – Descripción de los Monkeys

**Presentado Por:**

**Juan Carlos Arias Ramírez, 200327727**

## Que es el Simian Army?

El Simian Army consiste de servicios (Monkeys) en la nube para generar varios tipos de fallas, detectando condiciones anormales y probando nuestra capacidad para superar estas fallas.

Actualmente el Simian Army está compuesto por el Chaos Monkey, el Janitor Monkey, y el Conformity Monkey.

## Estrategias de Ataque del Chaos Monkey

Chaos Monkey es un servicio que identifica grupos de sistemas y termina aleatoriamente uno de los sistemas en un grupo. El servicio opera en un tiempo controlado (no se ejecuta los fines de semana y días festivos) y el intervalo (solo funciona durante el horario comercial). En la mayoría de los casos, hemos diseñado nuestras aplicaciones para que sigan funcionando cuando un compañero se desconecta, pero en esos casos especiales queremos asegurarnos de que haya personas disponibles para resolver y aprender de cualquier problema. Con esto en mente, Chaos Monkey solo funciona en horario comercial con la intención de que los ingenieros estén alertas y puedan responder.

La descripción de las configuraciones para este Monkey se puede consultar en el siguiente enlace:

https://github.com/Netflix/SimianArmy/wiki/Chaos-Settings

En el archivo chaos.properties también podemos encontrar las siguientes estrategias de ataque:

```
# Strategies
simianarmy.chaos.shutdowninstance.enabled = true
simianarmy.chaos.blockallnetworktraffic.enabled = false
simianarmy.chaos.burncpu.enabled = false
simianarmy.chaos.killprocesses.enabled = false
simianarmy.chaos.nullroute.enabled = false
simianarmy.chaos.failec2.enabled = false
simianarmy.chaos.faildns.enabled = false
simianarmy.chaos.faildynamodb.enabled = false
simianarmy.chaos.fails3.enabled = false
simianarmy.chaos.networkcorruption.enabled = false
simianarmy.chaos.networklatency.enabled = false
simianarmy.chaos.networkloss.enabled = false
```

La descripción de cada una de las estrategias se puede consultar en el siguiente link:

https://github.com/Netflix/SimianArmy/wiki/The-Chaos-Monkey-Army

### Shutdown instance *(Simius Mortus)*

Shuts down the instance using the EC2 API. The classic chaos monkey strategy.

Enable with: `simianarmy.chaos.shutdowninstance.enabled = true`

### Block all network traffic *(Simius Quies)*

This removes all security groups from the instance, and moves it into a security group that does not allow any access. Thus the instance is running, but cannot be reached via the network. This can only work on VPC instances.

Enable with: `simianarmy.chaos.blockallnetworktraffic.enabled = true`

Optionally configure the security group to use: `simianarmy.chaos.blockallnetworktraffic.group = blocked-network` (blocked-network is the default value)

The security group is auto-created; if the security group already exists, it will be used as-is. So if you wanted to leave e.g. port 22 open for diagnostic purposes, add it to the configured security group.

### Detach all EBS volumes *(Simius Amputa)*

This force-detaches all EBS volumes from the instance, simulating an EBS failure. Thus the instance is running, but EBS disk I/O will fail.

Enable with: `simianarmy.chaos.detachvolumes.enabled = true`

Warning: this monkey may cause data loss. Use this one carefully.

### SSH monkeys

The remaining monkeys all work by running scripts on the instance. To configure, you must set the SSH key to use to log in to the instance.

SSH username: `simianarmy.chaos.ssh.user = root` (The default is root) SSH key path: `simianarmy.chaos.ssh.key = ~/monkeyboy/.ssh/id_rsa` (path to your SSH key)

### Burn-CPU *(Simius Cogitarius)*

This monkey runs CPU intensive processes, simulating a noisy neighbor or a faulty CPU. The instance will effectively have a much slower CPU.

Enable with: `simianarmy.chaos.burncpu.enabled = true` Requires SSH to be configured.

### Burn-IO *(Simius Occupatus)*

This monkey runs disk intensive processes, simulating a noisy neighbor or a faulty disk. The instance will effectively have a much slower disk.

Enable with: `simianarmy.chaos.burnio.enabled = true` Requires SSH to be configured.

### Fill Disk *(Simius Plenus)*

This monkey writes a huge file to the root device, filling up the (typically relatively small) EC2 root disk.

Enable with: `simianarmy.chaos.filldisk.enabled = true` Requires SSH to be configured.

### Kill Processes *(Simius Delirius)*

This monkey kills any java or python programs it finds every second, simulating a faulty application, corrupted installation or faulty instance. The instance is fine, but the java/python application running on it will fail continuously.

Enable with: `simianarmy.chaos.killprocesses.enabled = true` Requires SSH to be configured.

### Null-Route *(Simius Desertus)*

This monkey null-routes the 10.0.0.0/8 network, which is used by the EC2 internal network. All EC2 <-> EC2 network traffic will fail.

Enable with: `simianarmy.chaos.nullroute.enabled = true` Requires SSH to be configured.

### Fail DNS *(Simius Nonomenius)*

This monkey uses iptables to block port 53 for TCP & UDP; those are the DNS traffic ports. This simulates a failure of your DNS servers.

Enable with: `simianarmy.chaos.faildns.enabled = true` Requires SSH to be configured.

### Fail EC2 API *(Simius Noneccius)*

This monkey puts dummy host entries into /etc/hosts, so that all EC2 API communication will fail. This simulates a failure of the EC2 control plane. Of course, if your application doesn't use the EC2 API from the instance, then it will be completely unaffected.

Enable with: `simianarmy.chaos.failec2.enabled = true` Requires SSH to be configured.

### Fail S3 API *(Simius Amnesius)*

This monkey puts dummy host entries into /etc/hosts, so that all S3 communication will fail. This simulates a failure of S3. Of course, if your application doesn't use S3, then it will be completely unaffected.

Enable with: `simianarmy.chaos.fails3.enabled = true` Requires SSH to be configured.

### Fail DynamoDB API *(Simius Nodynamus)*

This monkey puts dummy host entries into /etc/hosts, so that all DynamoDB communication will fail. This simulates a failure of DynamoDB. As with its monkey relatives, this will only affect instances that use DynamoDB.

Enable with: `simianarmy.chaos.faildynamodb.enabled = true` Requires SSH to be configured.

### Network Corruption *(Simius Politicus)*

This monkey uses the traffic shaping API to corupt a large fraction of network packets. This simulates degradation of the EC2 network.

Enable with: `simianarmy.chaos.networkcorruption.enabled = true` Requires SSH to be configured.

### Network Latency *(Simius Tardus)*

This monkey uses the traffic shaping API to introduce latency (1 second +- 50%) to all network packets. This simulates degradation of the EC2 network.

Enable with: `simianarmy.chaos.networklatency.enabled = true` Requires SSH to be configured.

### Network Loss *(Simius Perditus)*

This monkey uses the traffic shaping API to drop a fraction of all network packets. This simulates degradation of the EC2 network.

Enable with: `simianarmy.chaos.networkloss.enabled = true` Requires SSH to be configured.

**Reglas del Janitor Monkey**

Janitor Monkey es un servicio que se ejecuta en la nube de Amazon Web Services (AWS) en busca de recursos no utilizados para limpiar. El diseño de Janitor Monkey es lo suficientemente flexible como para permitir extenderlo a trabajar con otros proveedores de la nube y recursos de la nube. El servicio está configurado para ejecutarse, de forma predeterminada, en días de la semana no festivos a las 11 A. M. El programa puede ser fácilmente reconfigurado para adaptarse a las necesidades de su negocio.

Janitor Monkey determina si un recurso debe ser un candidato de limpieza aplicando un conjunto de reglas al respecto. Si alguna de las reglas determina que el recurso es un candidato de limpieza, Janitor Monkey marca el recurso y programa un tiempo para limpiarlo. El diseño de Janitor Monkey también hace que sea sencillo personalizar el conjunto de reglas o agregar nuevas.

Dado que siempre puede haber excepciones cuando desea mantener un recurso no utilizado más tiempo, antes de que Janitor Monkey elimine un recurso, el propietario del recurso recibirá una notificación con un número configurable de días antes del tiempo de limpieza. Esto es para evitar que Janitor Monkey elimine un recurso que todavía se necesita. El propietario del recurso puede marcar los recursos que desea mantener como excepciones y Janitor Monkey los dejará en paz.

¿Porque ejecutar el Janitor Monkey?

Una de las grandes ventajas de pasar de un centro de datos privado a la nube es que tiene acceso rápido y fácil a recursos nuevos casi ilimitados. Para enviar una nueva versión de la aplicación puede crear rápidamente un nuevo clúster, o cuando necesita más disco, solo adjunte un nuevo volumen, haga una copia de seguridad de sus datos usando un snapshot, o para probar una nueva idea solo cree nuevas instancias y comience a probarla. La desventaja de esta flexibilidad es que es bastante fácil perder la pista de los recursos de la nube que ya no se necesitan. Tal vez se olvidó de eliminar el clúster con la versión anterior de su aplicación u olvidó destruir el volumen cuando ya no necesitaba el disco adicional. Tomar snapshots es ideal para copias de seguridad, pero ¿realmente se necesitan los de hace 12 meses? No es solo el olvido lo que puede causar problemas, por ejemplo, los errores de red pueden hacer que su solicitud para eliminar un volumen no utilizado se pierda.

A menudo hay recursos no utilizados que cuestan dinero a los usuarios de la nube, y necesitábamos una solución para rectificar este problema. Los ingenieros diligentes pueden eliminar manualmente los recursos no utilizados, pero necesitamos una forma de detectarlos y limpiarlos automáticamente. La solución es Janitor Monkey.

¿Cómo hace la limpieza el Janitor Monkey?

Janitor Monkey trabaja en un proceso de "marcar, notificar y eliminar". Cuando Janitor Monkey marca un recurso como candidato de limpieza, programa un tiempo para eliminar el recurso. El tiempo de eliminación se especifica en la regla que marca el recurso. Todos los recursos están asociados con un correo electrónico del propietario, que se puede especificar como una etiqueta en el recurso o puede extender rápidamente Janitor Monkey para obtener la información de su sistema interno. La forma más simple es usar una dirección de correo electrónico predeterminada como por ejemplo la lista de correo electrónico de su equipo para todos los recursos. Puede configurar un número de días para especificar cuándo dejar que Janitor Monkey envíe notificaciones al propietario del recurso antes de la finalización programada. Por defecto, el número es 3, lo que significa que el propietario recibirá una notificación 3 días hábiles antes de la fecha de finalización. Durante el período de 3 días, el propietario del recurso puede decidir si el recurso está bien para eliminarlo. En caso de que un recurso deba retenerse por más tiempo, el propietario puede usar una interfaz REST simple para indicar que el recurso no ha sido limpiado por Janitor Monkey. El propietario siempre puede usar otra interfaz REST para eliminar la bandera y Janitor Monkey podrá administrar el recurso nuevamente. Cuando Janitor Monkey ve un recurso marcado como candidato a limpieza y el tiempo de finalización programado ya ha pasado, eliminará el recurso. El propietario del recurso también puede eliminar el recurso manualmente si desea liberar el recurso antes para ahorrar costos. Cuando el estado del recurso cambia lo que hace que el recurso no sea un candidato de limpieza, como por ejemplo un volumen separado de EBS se adjunta a una instancia, Janitor Monkey desmarca el recurso y no se producirá la terminación.

La descripción de las configuraciones y de las reglas para este Monkey se puede consultar en el siguiente enlace:

https://github.com/Netflix/SimianArmy/wiki/Janitor-Settings

Estas configuraciones se encuentran en el archivo janitor.properties.

A continuación se mencionan algunas de las reglas usadas por el Janitor Monkey:

simianarmy.janitor.rule.orphanedInstanceRule.enabled = true
simianarmy.janitor.rule.orphanedInstanceRule.instanceAgeThreshold = 2

```
simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithOwner = 3
simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithoutOwner = 8
simianarmy.janitor.rule.oldDetachedVolumeRule.enabled = true
simianarmy.janitor.rule.oldDetachedVolumeRule.detachDaysThreshold = 30
simianarmy.janitor.rule.oldDetachedVolumeRule.retentionDays = 7
simianarmy.janitor.rule.noGeneratedAMIRule.enabled = true
simianarmy.janitor.rule.noGeneratedAMIRule.ageThreshold = 30
simianarmy.janitor.rule.noGeneratedAMIRule.retentionDays = 7
simianarmy.janitor.rule.noGeneratedAMIRule.ownerEmail = null
simianarmy.janitor.Eureka.enabled = false
simianarmy.janitor.rule.oldEmptyASGRule.enabled = true
simianarmy.janitor.rule.oldEmptyASGRule.launchConfigAgeThreshold = 50
simianarmy.janitor.rule.oldEmptyASGRule.retentionDays = 10
simianarmy.janitor.rule.suspendedASGRule.enabled = true
simianarmy.janitor.rule.suspendedASGRule.suspensionAgeThreshold = 2
simianarmy.janitor.rule.suspendedASGRule.retentionDays = 5
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.enabled = true
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.ageThreshold = 4
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.retentionDays = 3
simianarmy.janitor.image.crawler.lookBackDays = 60
#simianarmy.janitor.image.ownerId = 1234567890
simianarmy.janitor.rule.unusedImageRule.enabled = false
simianarmy.janitor.rule.unusedImageRule.lastReferenceDaysThreshold = 45
simianarmy.janitor.rule.unusedImageRule.retentionDays = 3
```

## Setting rule of cleaning up instances not in auto scaling group

The properties below are used to configure the rule used to clean up orphaned instances that are not in any auto scaling group.

**simianarmy.janitor.rule.orphanedInstanceRule.enabled**

This property specifies whether Janitor monkey will clean up orphaned instances. If you do not want to clean up orphaned instances, you can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.orphanedInstanceRule.enabled = true
```

**simianarmy.janitor.rule.orphanedInstanceRule.instanceAgeThreshold**

An orphaned instance is marked as cleanup candidate if it has launched for more than the number of days specified in this property. The default value is 2, which means orphaned instance is marked as cleanup candidate and scheduled to be terminated if it has launched for more than 2 days.

```
simianarmy.janitor.rule.orphanedInstanceRule.instanceAgeThreshold = 2
```

**simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithOwner**

This property specifies the number of business days the instance is retained after a notification is sent about the termination when the instance has an owner. The default value is 3.

```
simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithOwner = 3
```

**simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithoutOwner**

This property specifies the number of business days the instance is retained after a notification is sent about the termination when the instance does not have an owner. The default value is 8.

```
simianarmy.janitor.rule.orphanedInstanceRule.retentionDaysWithoutOwner = 8
```

## Setting rule of cleaning up detached EBS volumes

The properties below are used to configure the rule used for cleaning up volumes that have been detached from instances for certain days. Since AWS does not provide a way to track when an EBS volume is detached, we provide a companion monkey called **Volume Tagging Monkey** for tracking this information. You can modify **volumeTagging.properties** file to enable it. Details can be found at Configure Volume Tagging Monkey.

**simianarmy.janitor.rule.oldDetachedVolumeRule.enabled**

This property specifies whether Janitor monkey will clean up detached volumes. If you do not want to clean up detached volumes, you can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.oldDetachedVolumeRule.enabled = true
```

**simianarmy.janitor.rule.oldDetachedVolumeRule.detachDaysThreshold**

A volume is considered a cleanup candidate after being detached for the number of days specified in this property. The default value is 30.

```
simianarmy.janitor.rule.oldDetachedVolumeRule.detachDaysThreshold = 30
```

**simianarmy.janitor.rule.oldDetachedVolumeRule.retentionDays**

This property specifies the number of business days the volume is retained after a notification is sent about the termination. The default value is 7.

```
simianarmy.janitor.rule.oldDetachedVolumeRule.retentionDays = 7
```

## Setting rule of cleaning up unreferenced snapshots

The properties below are used to configure the rule used for cleaning up snapshots that have no existing images generated from them and launched for certain days.

**simianarmy.janitor.rule.noGeneratedAMIRule.enabled**

This property specifies whether Janitor monkey will clean up unreferenced snapshots. You can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.noGeneratedAMIRule.enabled = true
```

**simianarmy.janitor.rule.noGeneratedAMIRule.ageThreshold**

A unreferenced snapshot is considered a cleanup candidate after launching for the number of days specified in this property. The default value is 30.

```
simianarmy.janitor.rule.noGeneratedAMIRule.ageThreshold = 30
```

**simianarmy.janitor.rule.noGeneratedAMIRule.retentionDays**

This property specifies the number of business days the snapshot is retained after a notification is sent about the termination. The default value is 7.

```
simianarmy.janitor.rule.noGeneratedAMIRule.retentionDays = 7
```

**simianarmy.janitor.rule.noGeneratedAMIRule.ownerEmail**

This property specifies an override owner email when cleaning EBS Snapshots with this rule. This will enable notification emails to be sent to a different email target rather than the resource owner. This could be useful for organizations that create a lot of snapshots as part of their build process and notifying owners is unnecessary.

The default value for this property is null meaning the resource owner will be notified and not the override value.

```
simianarmy.janitor.rule.noGeneratedAMIRule.ownerEmail = null
```

## Setting rule of cleaning up empty auto scaling groups

The properties below are used to configure the rule used for cleaning up auto scaling groups that have no active instances and the launch configuration is more than certain days old.

### simianarmy.janitor.Eureka.enabled

The property below specifies whether or not Eureka/Discovery is available for Janitor monkey to use. Discovery/Eureka is used in the rules for cleaning up auto scaling groups to determine if an auto scaling group has an 'active' instance, i.e. an instance that is registered and up in Discovery/Eureka. You should set this property to false if you do not have Discovery/Eureka set up in your environment. The default value of this property is false.

```
simianarmy.janitor.Eureka.enabled = false
```

### simianarmy.janitor.rule.oldEmptyASGRule.enabled

This property specifies whether Janitor monkey will clean up empty auto scaling groups. You can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.oldEmptyASGRule.enabled = true
```

### simianarmy.janitor.rule.oldEmptyASGRule.launchConfigAgeThreshold

An an auto-scaling group without active instances is considered a cleanup candidate when its launch configuration is older than the number of days specified in this property. The default value is 50.

```
simianarmy.janitor.rule.oldEmptyASGRule.launchConfigAgeThreshold = 50
```

### simianarmy.janitor.rule.oldEmptyASGRule.retentionDays

The number of business days an empty auto-scaling group is retained after a notification is sent for the termination. The default value is 10.

```
simianarmy.janitor.rule.oldEmptyASGRule.retentionDays = 10
```

## Setting rule of cleaning up suspended auto scaling groups

The properties below are used to configure the rule used for cleaning up auto-scaling groups that have no active instances and have been suspended from the associated ELB traffic for certain days.

**simianarmy.janitor.rule.suspendedASGRule.enabled**

This property specifies whether Janitor monkey will clean up suspended auto scaling groups. You can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.suspendedASGRule.enabled = true
```

**simianarmy.janitor.rule.suspendedASGRule.suspensionAgeThreshold**

An auto scaling group without active instances is considered a cleanup candidate when it has been suspended from the associated ELB traffic for the number of days specified in this property, the default value is 2.

```
simianarmy.janitor.rule.suspendedASGRule.suspensionAgeThreshold = 2
```

**simianarmy.janitor.rule.suspendedASGRule.retentionDays**

This property specifies the number of business days the auto scaling group is retained after a notification is sent for the termination. The default value is 5.

```
simianarmy.janitor.rule.suspendedASGRule.retentionDays = 5
```

## Setting rule of cleaning up launch configurations

The following properties are used by the Janitor rule for cleaning up launch configurations that are not used by any auto scaling group or instances and are older than certain days.

**simianarmy.janitor.rule.oldUnusedLaunchConfigRule.enabled**

This property specifies whether Janitor monkey will clean up unused launch configurations that are more than certain days old. You can set the property to false to disable the rule. The default value is true.

```
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.enabled = true
```

**simianarmy.janitor.rule.oldUnusedLaunchConfigRule.ageThreshold**

An unused launch configuration is considered a cleanup candidate when it is older than the number of days specified in the property below. The default value is 4.

```
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.ageThreshold = 4
```

**simianarmy.janitor.rule.oldUnusedLaunchConfigRule.retentionDays**

This property specifies the number of business days the launch configuration is kept after a notification is sent for the termination. The default value is 3.

```
simianarmy.janitor.rule.oldUnusedLaunchConfigRule.retentionDays = 3
```

## Setting rule of cleaning up images

The following properties are used by the Janitor rule for cleaning up images.

**simianarmy.janitor.image.crawler.lookBackDays**

The property below specifies the number of days to look back in the history when crawling the last reference information of the images. By default we look back up to 60 days.

```
simianarmy.janitor.image.crawler.lookBackDays = 60
```

**simianarmy.janitor.image.ownerId**

The property below specifies the owner id that images have for being managed by Janitor Monkey. If no owner id is set, all images under the AWS account are returned. By default the line is commented and no owner id is set.

```
#simianarmy.janitor.image.ownerId = 1234567890
```

**simianarmy.janitor.rule.unusedImageRule.enabled**

This rule is used by the Janitor rule for cleaning up images that have not been used by any instances and launch configurations, and and not used to create other images, in the last certain days. This rule is by default disabled, you need to have Edda running and enabled for using this rule since the image's history is needed to determine the last time when the images was referenced.

```
simianarmy.janitor.rule.unusedImageRule.enabled = false
```

**simianarmy.janitor.rule.unusedImageRule.lastReferenceDaysThreshold**

An unused image is considered a cleanup candidate when it is not referenced for than the number of days specified in the property below. The default value is 45.

```
simianarmy.janitor.rule.unusedImageRule.lastReferenceDaysThreshold = 45
```

**simianarmy.janitor.rule.unusedImageRule.retentionDays**

The property below specifies the number of business days the image is kept after a notification is sent for the termination. The default value is 3.

```
simianarmy.janitor.rule.unusedImageRule.retentionDays = 3
```

## Reglas del Conformity Monkey

Conformity Monkey es un servicio que se ejecuta en la nube de Amazon Web Services (AWS) buscando instancias que no se ajusten a las reglas predefinidas para las mejores prácticas. El diseño de Conformity Monkey es lo suficientemente flexible como para permitir extenderlo y trabajar con otros proveedores de nube y recursos de la nube. Por defecto, la verificación de conformidad se realiza cada hora. El programa puede ser fácilmente reconfigurado para adaptarse a las necesidades de su negocio.

Conformity Monkey determina si una instancia no es conforme aplicando un conjunto de reglas sobre ella. Si alguna de las reglas determina que la instancia no se está cumpliendo, el monkey envía una notificación por correo electrónico al propietario de la instancia. Proporcionamos una colección de reglas de conformidad en la versión de fuente abierta que actualmente se usan en Netflix y que se consideran lo suficientemente generales como para ser utilizadas por la mayoría de los usuarios. El diseño de Conformity Monkey también facilita la personalización de reglas o la adición de nuevas reglas.

Puede haber excepciones cuando desea ignorar las advertencias de una regla de conformidad específica para algunas aplicaciones. Por ejemplo, un grupo de seguridad para abrir un puerto específico probablemente no es necesario por instancias de algunas aplicaciones. Le permitimos personalizar el conjunto de reglas de conformidad que se aplicarán a un clúster de instancias excluyendo las innecesarias. O puede optar por excluirse por completo de un grupo específico de Conformity Monkey, por lo que no se envía ninguna notificación.

### ¿Porque ejecutar el Conformity Monkey?

Cloud Computing facilita mucho el lanzamiento de nuevas aplicaciones o el inicio de nuevas instancias. En Netflix, los ingenieros pueden lanzar fácilmente una nueva aplicación en Asgard con unos pocos clicks. Con esta libertad, a veces hay consecuencias en las que las aplicaciones o instancias lanzadas pueden no seguir algunas de las mejores prácticas, tal vez el ingeniero no conocía las mejores prácticas o simplemente se olvidaba de ellas. Por ejemplo, algunos grupos de seguridad necesarios pueden faltar en las instancias y pueden causar brechas de seguridad. O tal vez no se haya definido una URL de verificación de estado para las instancias en Eureka, lo que daría como resultado la detección automática de fallas y la desactivación del failover.

### ¿Cómo trabaja el Conformity Monkey?

Conformity Monkey funciona en dos etapas: marcar y notificar. En primer lugar, Conformity Monkey recorre todos los grupos de escalado automático en su nube y aplica el conjunto especificado de reglas de conformidad a las instancias de cada grupo. Si alguna regla de conformidad determina que una instancia no se ajusta, el grupo de escalado automático se marca como no conforme y se registran las instancias que violan la regla. Cada grupo de escalado automático está asociado con un correo electrónico de propietario, que se puede obtener de un sistema interno o se puede configurar en un archivo de configuración. La forma más simple es usar una dirección de correo electrónico predeterminada como por ejemplo la lista de correo electrónico de su equipo para todos los grupos de autoescalamiento. Conformity Monkey envía al propietario una notificación por correo electrónico sobre los grupos no conformes, con los detalles de la regla de conformidad rota y las instancias que no superaron la verificación de conformidad. Los propietarios de la aplicación pueden entonces tomar las medidas necesarias para arreglar las instancias fallidas o excluir la regla de conformidad si creen que

la verificación de conformidad no es necesaria para la aplicación. Le permitimos configurar diferentes frecuencias para la verificación de conformidad y la notificación. Por ejemplo, en Netflix, la verificación de conformidad se realiza cada hora, y la notificación solo se envía una vez al día al mediodía. Esto es para reducir la cantidad de correos electrónicos que las personas reciben sobre la misma advertencia de conformidad. El resultado en tiempo real de la verificación de conformidad para cada grupo de escalado automático se muestra en una UI separada.

La descripción de las configuraciones y de las reglas para este Monkey se puede consultar en el siguiente enlace:

https://github.com/Netflix/SimianArmy/wiki/Conformity-Settings

Estas configuraciones se encuentran en el archivo conformity.properties.

A continuación se mencionan algunas de las reglas usadas por el Conformity Monkey:

simianarmy.conformity.rule.SameZonesInElbAndAsg.enabled = true
simianarmy.conformity.rule.InstanceInSecurityGroup.enabled = true
simianarmy.conformity.rule.InstanceInSecurityGroup.requiredSecurityGroups = foo, bar
simianarmy.conformity.rule.InstanceTooOld.enabled = true
simianarmy.conformity.rule.InstanceTooOld.instanceAgeThreshold = 180
simianarmy.conformity.rule.InstanceInVPC.enabled = true
simianarmy.conformity.rule.InstanceHasStatusUrl.enabled = true
simianarmy.conformity.rule.InstanceHasHealthCheckUrl.enabled = true

**simianarmy.conformity.rule.SameZonesInElbAndAsg.enabled**

This property is used to enable the conformity rule to check whether there is mismatch of availability zones between any auto scaling group and its elastic load balancers in a cluster.

```
simianarmy.conformity.rule.SameZonesInElbAndAsg.enabled = true
```

**simianarmy.conformity.rule.InstanceInSecurityGroup.enabled**

This property is used to enable the conformity rule to check whether all instances in a cluster are in required security groups.

```
simianarmy.conformity.rule.InstanceInSecurityGroup.enabled = true
```

**simianarmy.conformity.rule.InstanceInSecurityGroup.requiredSecurityGroups**

This property specifies the required security groups in the InstanceInSecurityGroup conformity rule.

```
simianarmy.conformity.rule.InstanceInSecurityGroup.requiredSecurityGroups = foo, bar
```

**simianarmy.conformity.rule.InstanceTooOld.enabled**

This property is used to enable the conformity rule to check whether there is any instance that is older than certain days.

```
simianarmy.conformity.rule.InstanceTooOld.enabled = true
```

**simianarmy.conformity.rule.InstanceTooOld.instanceAgeThreshold**

This property specifies the number of days used in the InstanceInSecurityGroup, any instance that is old than this number of days is consider nonconforming.

```
simianarmy.conformity.rule.InstanceTooOld.instanceAgeThreshold = 180
```

**simianarmy.conformity.rule.InstanceInVPC.enabled**

This property is used to enable the conformity rule to check whether your instances are in an Amazon Virtual Private Cloud (VPC).

```
simianarmy.conformity.rule.InstanceInVPC.enabled = true
```

**simianarmy.conformity.rule.InstanceHasStatusUrl.enabled**

This property is used to enable the conformity rule to check whether all instances in the cluster have a status url defined in Discovery/Eureka. The rule is added to Conformity Monkey only when Eureka is enabled also.

```
simianarmy.conformity.rule.InstanceHasStatusUrl.enabled = true
```

**simianarmy.conformity.rule.InstanceHasHealthCheckUrl.enabled**

This property is used to enable the conformity rule to check whether all instances in the cluster have a health check url defined in Discovery/Eureka. The rule is added to Conformity Monkey only when Eureka is enabled also.

```
simianarmy.conformity.rule.InstanceHasHealthCheckUrl.enabled = true
```