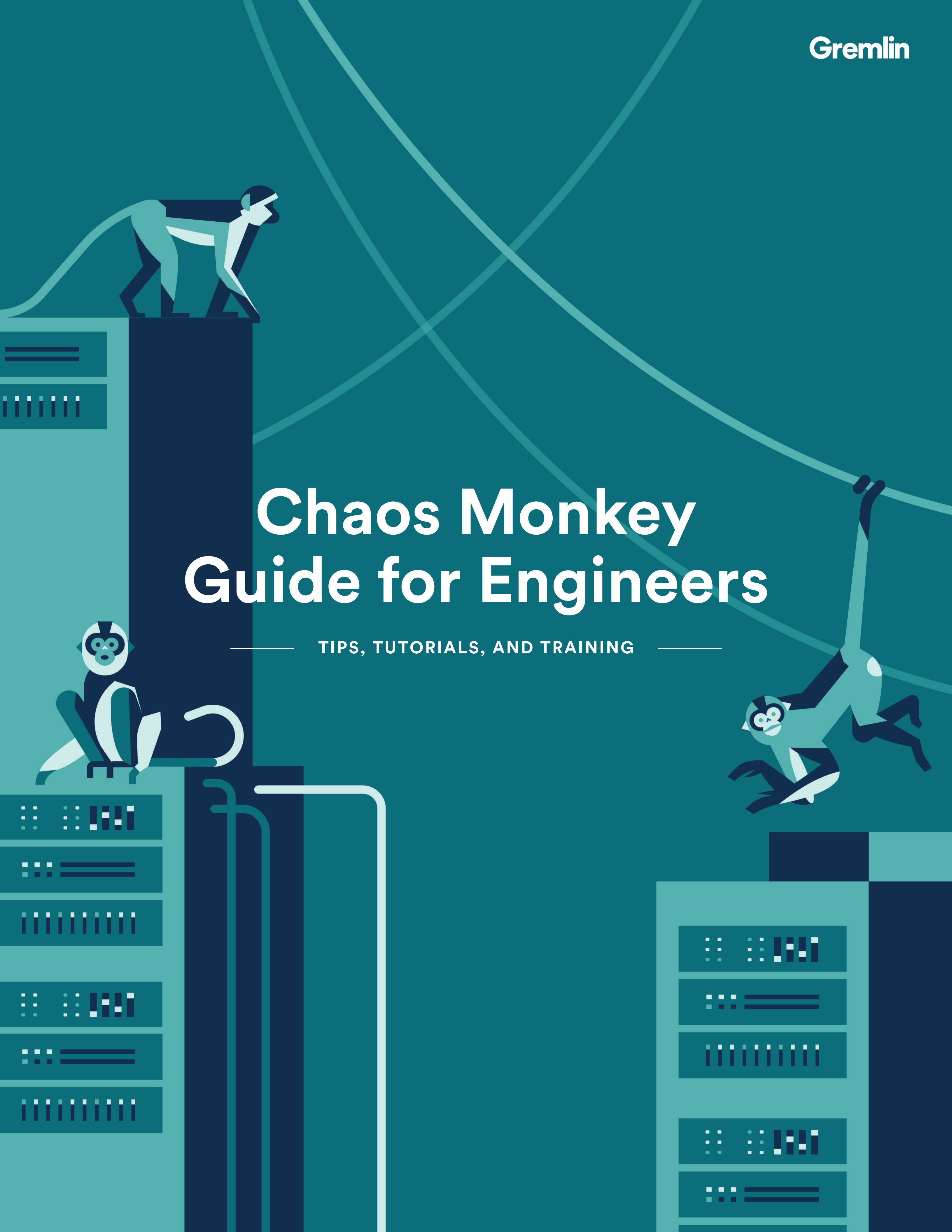
Chaos Monkey Guide for Engineers

TIPS, TUTORIALS, AND TRAINING

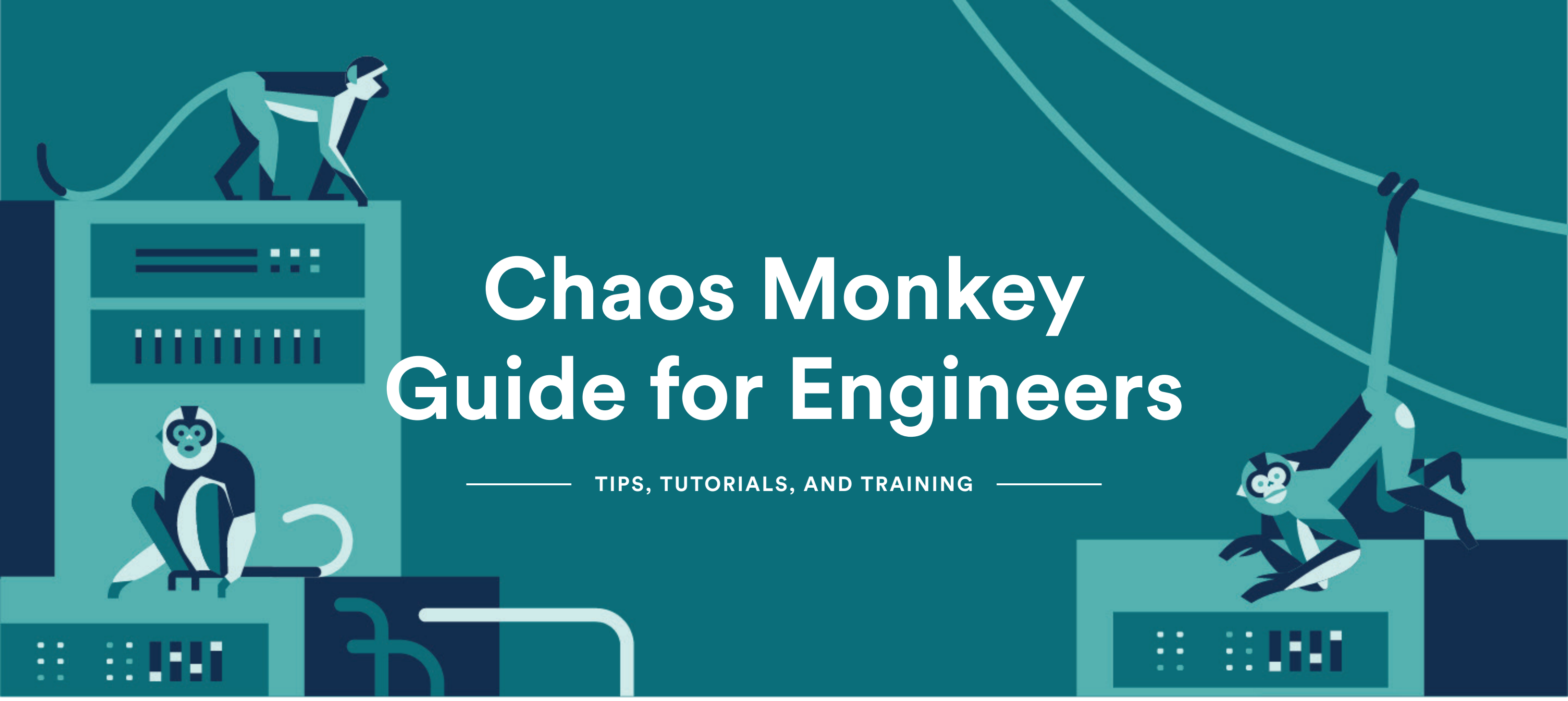


Chaos Monkey Guide for Engineers

TIPS, TUTORIALS, AND TRAINING

Table of Contents

Table of Contents



Chaos Monkey Guide for Engineers

TIPS, TUTORIALS, AND TRAINING

In 2010 Netflix announced the existence and success of their custom resiliency tool called *Chaos Monkey*.

In 2010, Netflix decided to move their systems to the cloud. In this new environment, hosts could be terminated and replaced at any time, which meant their services needed to prepare for this constraint. By pseudo-randomly rebooting their own hosts, they could suss out any weaknesses and validate that their automated remediation worked correctly. This also helped find “stateful” services, which relied on host resources (such as a local cache and database), as opposed to stateless services, which store such things on a remote host.

Netflix designed Chaos Monkey to test system stability by enforcing failures via the pseudo-random termination of instances and services within Netflix’s architecture. Following their migration to the cloud, Netflix’s service was newly reliant upon Amazon Web Services and needed a technology that could show them how their system responded when critical components of their production service infrastructure were taken down. Intentionally causing this single failure would suss out any weaknesses in their systems and guide them towards automated solutions that gracefully handle future failures of this sort.

CHAOS ENGINEERING IS

> “the discipline of experimenting on a distributed system in order to build confidence in the system’s capability to withstand turbulent conditions in production.”

Chaos Monkey helped jumpstart Chaos Engineering as a new engineering practice. Chaos Engineering is “the discipline of experimenting on a distributed system in order to build confidence in the system’s capability to withstand turbulent conditions in production.” By proactively testing how a system responds to failure conditions, you can identify and fix failures before they become public facing outages. Chaos Engineering lets you compare what you think will happen with what is actually happening in your systems. By performing the smallest possible experiments you can measure, you're able to "break things on purpose" in order to learn how to build more resilient systems.

In 2011, Netflix announced the evolution of Chaos Monkey with a series of additional tools known as The Simian Army. Inspired by the success of their original Chaos Monkey tool aimed at randomly disabling production instances and services, the engineering team developed additional "simians" built to cause other types of failure and induce abnormal system conditions. For example, the Latency Monkey tool introduces artificial delays in RESTful client-server communication, allowing the team at Netflix to simulate service unavailability without actually taking down said service. This guide will cover all the details of these tools in The Simian Army chapter.

What Is This Guide?

The Chaos Monkey Guide for Engineers is a full how-to of Chaos Monkey, including what it is, its origin story, its pros and cons, its relation to the broader topic of Chaos Engineering, and much more. We've also included step-by-step technical tutorials for getting started with Chaos Monkey, along with advanced engineering tips and guides for those looking to go beyond the basics. The Simian Army section explores all the additional tools created after Chaos Monkey.

This guide also includes resources, tutorials, and downloads for engineers seeking to improve their own Chaos Engineering practices. In fact, our alternative technologies chapter goes above and beyond by examining a curated list of the best alternatives to Chaos Monkey – we dig into everything from Azure and Docker to Kubernetes and VMware!

Who Is This Guide For?

We've created this guide primarily for engineers and other enterprise technologists who are looking for the ultimate resource on Chaos Monkey, as a way to get started with Chaos Engineering. We want to help readers see how Chaos Monkey fits into the whole realm of Chaos Engineering practices.

Why Did We Create This Guide?

Our goal here at Gremlin is to empower engineering teams to build more resilient systems through thoughtful Chaos Engineering. We're on a constant quest to promote the Chaos Community through frequent conferences & meetups, in-depth talks, detailed tutorials, and the ever-growing list of Chaos Engineering Slack channels.

While Chaos Engineering extends well beyond the scope of one single technique or idea, Chaos Monkey is the most well-known tool for running Chaos Experiments and is a common starting place for engineers getting started with the discipline.

The Pros and Cons of Chaos Monkey

Chaos Monkey is designed to induce one specific type of failure. It randomly shuts down instances in order to simulate random server failure.

Pros of Chaos Monkey

PREPARES YOU FOR RANDOM FAILURES

Chaos Monkey allows for planned instance failures when you and your team are best-prepared to handle them. You can schedule terminations so they occur based on a configurable mean number of days and during a given time period each day.

ENCOURAGES DISTRIBUTION

As Netflix learned all too well in 2008 prior to developing Chaos Monkey, a vertically-stacked architecture is dangerous and prone to single points of failure. Conversely, a distributed architecture that Chaos Engineering practices and tools like Chaos Monkey encourage is inherently more resilient, so long as you proactively "break things on purpose" in an effort to learn.

ENCOURAGES REDUNDANCY

Part and parcel of a distributed architecture, redundancy is another major benefit to smart Chaos Engineering practices. If a single service or instance is brought down unexpectedly, a redundant backup may save the day.

DISCOURAGES WASTEFUL DEPENDENCIES

Chaos Engineering best practices emphasize the importance of separating the wheat from the chaff by eliminating all unnecessary dependencies and allowing the system to remain functional with the absolute minimal components and services.

DISCOVERING IMPROVEMENTS

Performing Chaos Experiments can often shed light on previously unknown improvements and workarounds. ("Turns out, even with our core XYZ service offline, we're still going. Awesome!")

BUILT INTO SPINNAKER

If your architecture already relies on Spinnaker, getting Chaos Monkey up and running is a breeze.

Cons of Chaos Monkey

REQUIRES SPINNAKER

As discussed in The Origin of Chaos Monkey, Chaos Monkey does not support deployments that are managed by anything other than Spinnaker.

REQUIRES MYSQL

Chaos Monkey also requires the use of MySQL 5.X, as discussed in more detail in the Chaos Monkey Tutorial chapter.

LIMITED FAILURE MODE

Chaos Monkey's limited scope means it injects one type of failure – causing pseudo-random instance failure. Thoughtful Chaos Engineering is about looking at an application's future, toward unknowable and unpredictable failures, beyond those of a single AWS instance. Chaos Monkey only handles a tiny subset of the "long tail" failures that software will experience during its life cycle. Check out the Chaos Monkey Alternatives chapter for more information.

LACK OF COORDINATION

While Chaos Monkey can terminate instances and cause failures, it lacks much semblance of coordination. Since Chaos Monkey is an open-source tool that was built by and for Netflix, it's left to you as the end-user to inject your own system-specific logic. Bringing down an instance is great and all, but knowing how to coordinate and act on that information is critical.

NO RECOVERY CAPABILITIES

A big reason why Chaos Engineering encourages performing the smallest possible experiments is so any repercussions are somewhat contained – if something goes awry, it's ideal to have a safety net or the ability to abort the experiment. Unfortunately, while Chaos Monkey doesn't include such safety features, many other tools and services have these capabilities, including Gremlin's Halt All button, which immediately stops all active experiments.

LIMITED HELPER TOOLS

As with most open source projects, Chaos Monkey is entirely executed through the command line, scripts, and configuration files. If your team wants an interface, it's up to you to build it.

NO USER INTERFACE

By itself, Chaos Monkey fails to provide many useful functions such as auditing, outage checking, termination tracking, and so forth. Spinnaker supports a framework for creating your own Chaos Monkey auditing through its Echo events microservice, but you'll generally be required to either integrate with Netflix's existing software or to create your own custom tools in order to get much info out of Chaos Monkey.

THE ORIGIN OF CHAOS MONKEY

Why Netflix Needed to Create Failure

In this chapter we'll take a deep dive into the origins and history of Chaos Monkey, how Netflix streaming services emerged, and why Netflix needed to create failure within their systems to improve their service and customer experiences. We'll also provide a brief overview of The Simian Army and its relation to the original Chaos Monkey technology. Finally, we'll jump into the present and future of Chaos Monkey, dig into the creation and implementation of Failure Injection Testing at Netflix, and discuss the potential issues and limitations presented by Chaos Monkey's reliance on Spinnaker.

The History of Netflix Streaming

Netflix launched their streaming service in early 2007, as a free addon for their existing DVD-by-mail subscribers. While their initial streaming library contained only around 1,000 titles at launch, the popularity and demand continued to rise, and Netflix kept adding to their streaming library, reaching over 12,000 titles by June 2009.

Netflix's streaming service was initially built by Netflix engineers on top of Microsoft software and housed within vertically scaled server racks. However, this single point of failure came back to bite them in August 2008, when a major database corruption resulted in a three-day downtime during which DVDs couldn't be shipped to customers. Following this event, Netflix engineers began migrating the entire Netflix stack away from a monolithic architecture, and into a distributed cloud architecture, deployed on Amazon Web Services.

This major shift toward a distributed architecture of hundreds of microservices presented a great deal of additional complexity. This level of intricacy and interconnectedness in a distributed system created something that was intractable and required a new approach to prevent seemingly random outages. But by using proper Chaos Engineering techniques, starting first with Chaos Monkey and evolving into more sophisticated tools like FIT, Netflix was able to engineer a resilient architecture.

Netflix's move toward a horizontally scaled software stack required systems that were much more reliable and fault tolerant. One of the most critical lessons was that "the best way to avoid failure is to fail constantly.". The engineering team needed a tool that could proactively inject failure into the system. This would show the team how the system behaved under abnormal conditions, and would teach them how to alter the system so other services could easily tolerate future, unplanned failures. Thus, the Netflix team began their journey into Chaos.

The Simian Army

The Simian Army is a suite of failure injection tools created by Netflix that shore up some of the limitations of Chaos Monkey's scope. Check out the Simian Army - Overview and Resources chapter for all the details on what the Simian Army is, why it was created, the tools that make up the Army, the strategies used to perform various Chaos Experiments, and a tutorial to help you install and begin using the Simian Army tools.

Chaos Monkey Today

Chaos Monkey 2.0 was announced and publicly released on GitHub in late 2016. The new version includes a handful of major feature changes and additions.

SPINNAKER REQUIREMENT

Spinnaker is an open-source, multi-cloud continuous delivery platform developed by Netflix, which allows for automated deployments across multiple cloud providers like AWS, Azure, Kubernetes, and a few more. One major drawback of using Chaos Monkey is that it forces you and your organization to build atop Spinnaker's CD architecture. If you need some guidance on that, check out our Spinnaker deployment tutorials.

IMPROVED SCHEDULING

Instance termination schedules are no longer determined by probabilistic algorithms, but are instead based on the mean time between terminations. Check out How to Schedule Chaos Monkey Terminations for technical instructions.

TRACKERS

Trackers are Go language objects that report instance terminations to external services.

LOSS OF ADDITIONAL CAPABILITIES

Prior to 2.0, Chaos Monkey was capable of performing additional actions beyond just terminating instances. With version 2.0, those capabilities have been removed and moved to other Simian Army tools.

Failure Injection Testing

In October 2014, dissatisfied with the lack of control introduced when unleashing some of The Simian Army tools, Netflix introduced a solution they called Failure Injection Testing (FIT). Built by a small team of Netflix engineers – including Gremlin Co-Founder and CEO Kolton Andrus – FIT added dimensions to the failure injection process, allowing Netflix to more precisely determine what was failing and which components that failure impacted.

FIT works by first pushing failure simulation metadata to Zuul, which is an edge service developed by Netflix. Zuul handles all requests from devices and applications that utilize the back end of Netflix's streaming service. As of version 2.0, Zuul can handle dynamic routing, monitoring, security, resiliency, load balancing, connection pooling, and more. The core functionality of Zuul's business logic comes from Filters, which behave like simple **pass/fail** tests applied to each request and determine if a given action should be performed for that request. A filter can handle actions such as adding debug logging, determining if a response should be GZipped, or attaching injected failure, as in the case of FIT.

The introduction of FIT into Netflix's failure injection strategy was a good move toward better, modern-day Chaos Engineering practices. Since FIT is a service unto itself, it allowed failure to be injected by a variety of teams, who could then perform proactive Chaos Experiments with greater precision. This allowed Netflix to truly emphasize a core discipline of Chaos Engineering, knowing they were testing for failure in every nook and cranny, proving confidence that their systems were resilient to truly *unexpected* failures.

Unlike Chaos Monkey, tools like FIT and Gremlin are able to test for a wide range of failure states beyond simple instance destruction. In addition to killing instances, Gremlin can fill available disk space, hog CPU and memory, overload IO, perform advanced network traffic manipulation, terminate processes, and much more.

Chaos Monkey and Spinnaker

As discussed above and later in our Spinnaker Quick Start guide, Chaos Monkey can **only** be used to terminate instances within an application managed by Spinnaker.

This requirement is not a problem for Netflix or those other companies (such as Waze) that using Spinnaker to great success. However, limiting your Chaos Engineering tools and practices to just Chaos Monkey also means limiting yourself to only Spinnaker as your continuous delivery and deployment solution. This is a great solution if you're looking to tightly integrate with all the tools Spinnaker brings with it. On the other hand, if you're looking to expand out into other tools this may present a number of potential issues:

SETUP AND PROPAGATION

Spinnaker requires quite a bit of investment in server setup and propagation. As you may notice in even the streamlined, provider-specific tutorials found later in this guide, getting Spinnaker up and running on a production environment takes a lot of time (and a hefty number of CPU cycles).

LIMITED DOCUMENTATION

Spinnaker's official documentation is rather limited and somewhat outdated in certain areas.

PROVIDER SUPPORT

Spinnaker currently supports most of the big name cloud providers, but if your use case requires a provider outside of this list you're out of luck (or will need to develop your own CloudDriver).

CHAOS MONKEY TUTORIAL

A Step-by-Step Guide to Creating Failure on AWS

This chapter will provide a step-by-step guide for setting up and using Chaos Monkey with AWS. We also examine a handful of scenarios in which Chaos Monkey is not always the most relevant solution for Chaos Engineering implementation, due to its Spinnaker requirements and limited scope of only handling instance terminations.

How to Quickly Deploy Spinnaker for Chaos Monkey

Modern Chaos Monkey **requires** the use of Spinnaker, which is an open-source, multi-cloud continuous delivery platform developed by Netflix. Spinnaker allows for automated deployments across multiple cloud platforms (such as AWS, Azure, Google Cloud Platform, and more). Spinnaker can also be used to deploy across multiple accounts and regions, often using **pipelines** that define a series of events that should occur every time a new version is released. Spinnaker is a powerful tool, but since both Spinnaker and Chaos Monkey were developed by and for Netflix's own architecture, you'll need to do the extra legwork to configure Spinnaker to work within your application and infrastructure.

That said, in this first section we'll explore the fastest and simplest way to get Spinnaker up and running, which will then allow you to move onto installing and then using.

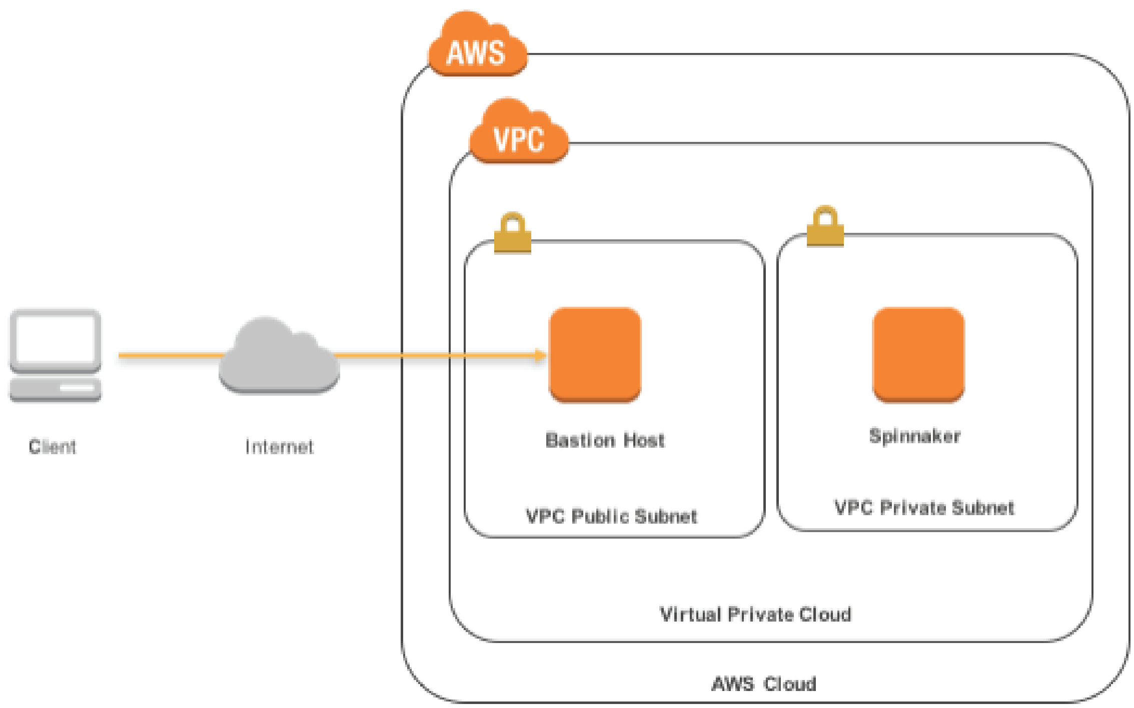
We'll be deploying Spinnaker on AWS, and the easiest method for

Looking to Deploy Spinnaker In Another Environment?

If you're looking for the utmost control over your Spinnaker deployment you should check out our [How to Deploy a Spinnaker Stack for Chaos Monkey][#spinnaker-manual] guide, which provides a step-by-step tutorial for setting up Halyard and Spinnaker on a local or virtual machine of your choice.

doing so is to use the CloudFormation Quick Start template.

The AWS Spinnaker Quick Start will create a simple architecture for you containing two subnets (one public and one private) in a Virtual Private Cloud (VPC). The public subnet contains a Bastion host instance designed to be strictly accessible, with just port 22 open for SSH access. The Bastion host will then allow a pass through connection to the private subnet that is running Spinnaker.



*AWS Spinnaker Quick Start Architecture - **Courtesy of AWS****

This quick start process will take about 10 - 15 minutes and is mostly automated.

Creating the Spinnaker Stack

1. *(Optional)* If necessary, visit https://aws.amazon.com/ to sign up for or login to your AWS account.

2. *(Optional)* You'll need at least one AWS EC2 Key Pair for securely connecting via SSH.

1. If you don't have a KeyPair already start by opening the AWS Console and navigate to **EC2 > NETWORK & SECURITY > Key Pairs**.

2. Click **Create Key Pair** and enter an identifying name in the **Key pair** name field.

3. Click **Create** to download the private `.pem` key file to your local system.

4. Save this key to an appropriate location (typically your local user `~/.ssh` directory).

3. After you've signed into the AWS console visit this page, which should load the `quickstart-spinnakercf.template.`

4. Click **Next**.

5. *(Optional)* If you haven't already done so, you'll need to create at least one AWS Access Key.

6. Select the **KeyName** of the key pair you previously created.

7. Input a secure password in the **Password** field.

8. *(Optional)* Modify the IP address range in the **SSHLocation** field to indicate what IP range is allowed to SSH into the Bastion host. For example, if your public IP address is 1.2.3.4 you might enter 1.2.3.4/32 into this field. If you aren't sure, you can enter 0.0.0.0/0 to allow any IP address to connect, though this is obviously less secure.

9. Click **Next**.

10. (Optional) Select an **IAM Role** with proper CloudFormation permissions necessary to deploy a stack. If you aren't sure, leave this blank and deployment will use your account's permissions.

11. Modify any other fields on this screen you wish, then click **Next** to continue.

If your AWS account already contains the BaseIAMRole AWS::IAM::Role you may have to delete it before this template will succeed.

12. Check the **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. checkbox and click **Create** to generate the stack.

13. Once the Spinnaker stack has a CREATE_COMPLETE **Status**, select the **Outputs** tab, which has some auto-generated strings you'll need to paste in your terminal in the next section.

Connecting to the Bastion Host

1. Copy the **Value** of the **SSHString1** field from the stack **Outputs** tab above.

2. Execute the **SSHString1** value in your terminal and enter yes when prompted to continue connecting to this host.

```
ssh -A -L 9000:localhost:9000 -L 8084:localhost:8084 -L
8087:localhost:8087 ec2-user@54.244.189.78
```

3. You should now be connected as the ec2-user to the Bastion instance. Before you can connect to the Spinnaker instance you'll probably need to copy your .pem file to the Spinnaker instance's ~/.ssh directory.

   - Once the key is copied, make sure you set proper permissions otherwise SSH will complain.

```
chmod 400 ~/.ssh/my_key.pem
```

Permission denied (publickey).

If you received a permission denied SSH error you may have forgotten to place your .pem private key file that you downloaded from the AWS EC2 Key Pair creation page. Make sure it is located in your ~/.ssh user directory. Otherwise you can specify the key by adding an optional -i <identify_file_path> flag, indicating the path to the .pem file.

Connecting to the Spinnaker Host

1. To connect to the Spinnaker instance copy and paste the **SSHString2 Value** into the terminal.

```
ssh -L 9000:localhost:9000 -L 8084:localhost:8084 -L
8087:localhost:8087 ubuntu@10.100.10.167 -i ~/.ssh/my_
key.pem
```

Permission denied (publickey).

Upon connecting to the Spinnaker instance you may see a message indicating the system needs to be restarted. You can do this through the AWS EC2 console, or just enter the `sudo reboot` command in the terminal, then reconnect after a few moments.

2. You should now be connected to the `SpinnakerWebServer`!

Configuring Spinnaker

The Spinnaker architecture is composed of a collection of microservices that each handle various aspects of the entire service. For example, Deck is the web interface you'll spend most time interacting with, Gate is the API gateway that handles most communication between microservices, and CloudDriver is the service that communicates and configures all cloud providers Spinnaker is working with.

Since so much of Spinnaker is blown out into smaller microservices, configuring Spinnaker can require messing with a few different files. If there's an issue you'll likely have to look through individual logs for each different service, depending on the problem.

Spinnaker is configured through `/opt/spinnaker/config/spinnaker.yml` file. However, this file will be overwritten by Halyard or other changes, so for user-generated configuration you should actually modify the `/opt/spinnaker/config/spinnaker-local.yml` file. Here's a basic example of what that file looks like.

```
# /opt/spinnaker/config/spinnaker-local.yml
global:
  spinnaker:
    timezone: 'America/Los_Angeles'
providers:
  aws:
    # For more information on configuring Amazon Web Services (aws), see
    # http://www.spinnaker.io/v1.0/docs/target-deployment-setup#section-amazon-web-services-setup
```

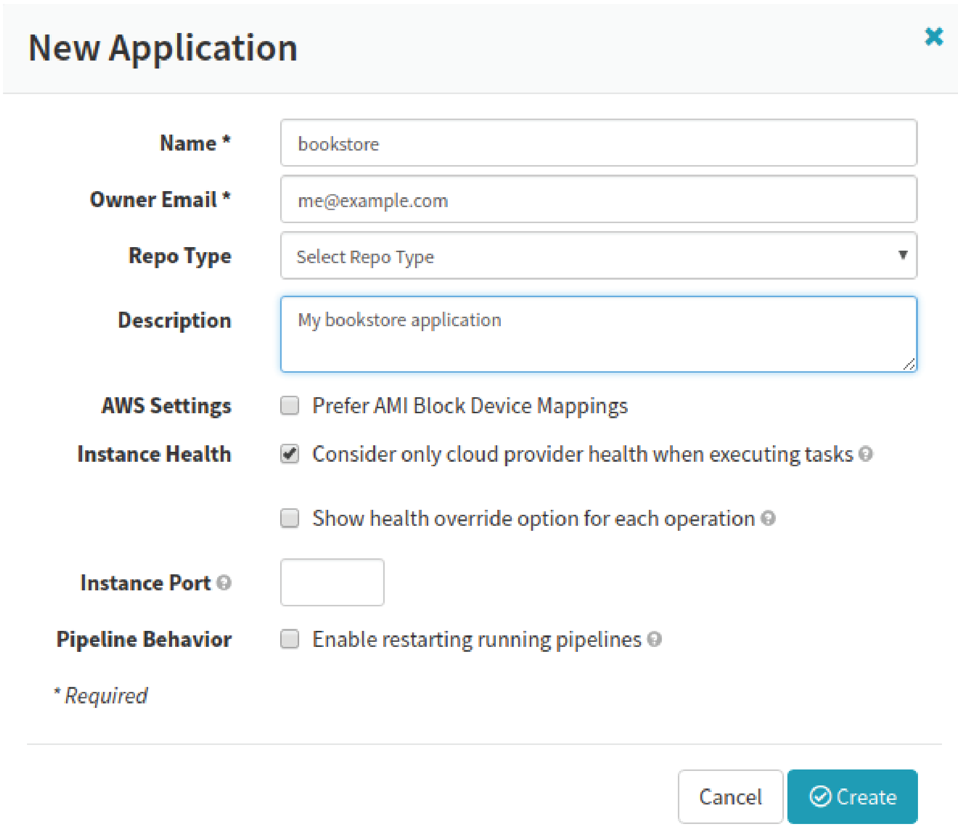
```
enabled: ${SPINNAKER_AWS_ENABLED:false}
defaultRegion: ${SPINNAKER_AWS_DEFAULT_REGION:us-west-2}
defaultIAMRole: Spinnaker-BaseIAMRole-GAT2AISI7TMJ
primaryCredentials:
  name: default
  # Store actual credentials in $HOME/.aws/credentials. See spinnaker.yml
  # for more information, including alternatives.

#  will be interpolated with the aws account name (e.g. "my-aws-account-keypair").
defaultKeyPairTemplate: "-keypair"

# ...
```

Standalone Spinnaker installations (such as the one created via the AWS Spinnaker Quick Start) are configured directly through the `spinnaker.yml` and `spinnaker-local.yml` override configuration files.

Creating an Application

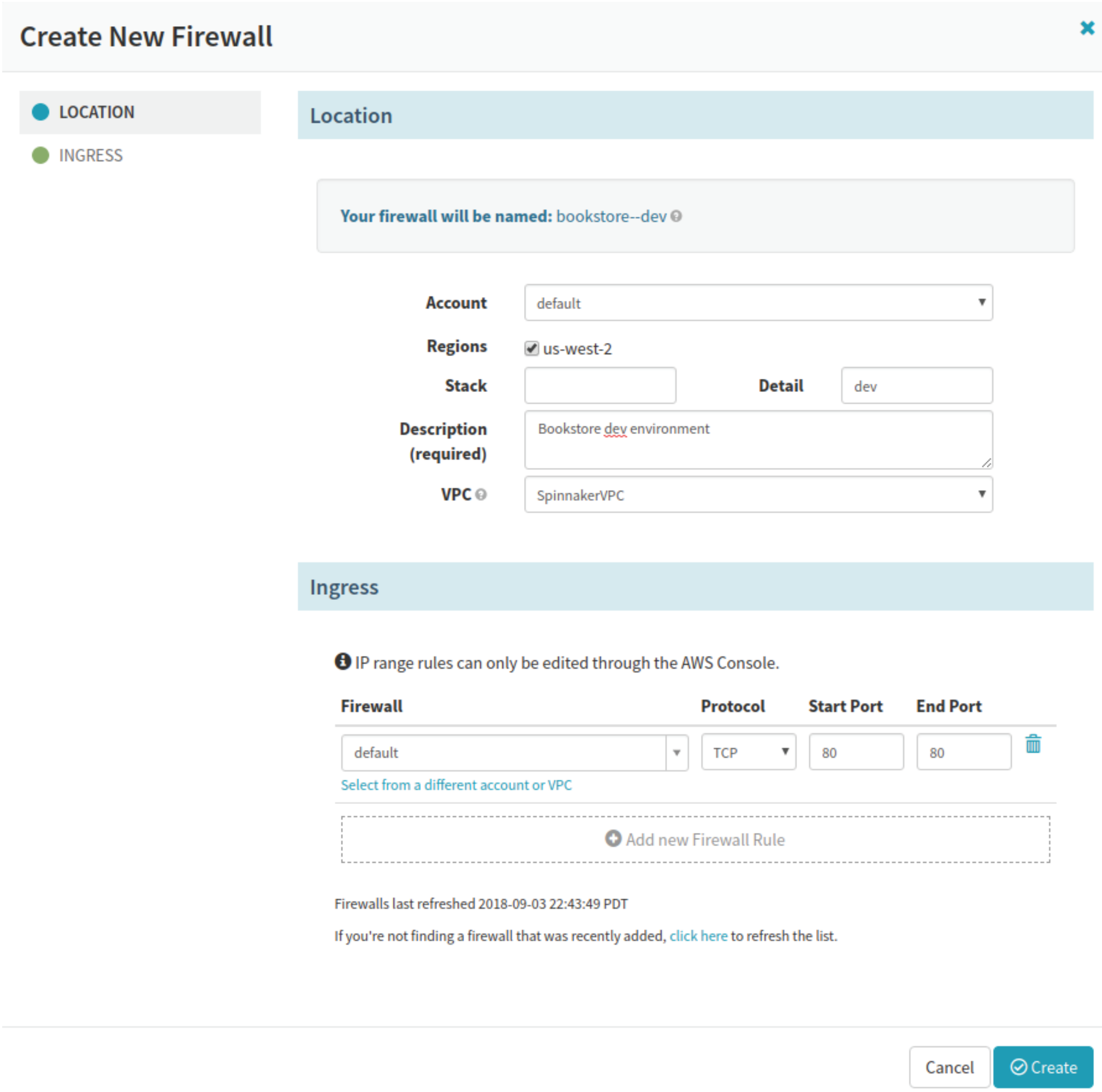
In this section we'll manually create a Spinnaker **application** containing a pipeline that first *bakes* a virtual machine image and then *deploys* that image to a cluster.

1. Open the Spinnaker web UI (**Deck**) and click **Actions > Create** Application.
2. Input `bookstore` in the **Name** field.
3. Input your own email address in the **Owner Email** field.
4. *(Optional)* If you've enabled Chaos Monkey in Spinnaker you can opt to enable Chaos Monkey by checking the **Chaos Monkey > Enabled** box.
5. Input `My bookstore application` in the **Description** field.
6. Under **Instance Health**, tick the **Consider only cloud provider health when executing tasks** checkbox.
7. Click **Create** to add your new application.



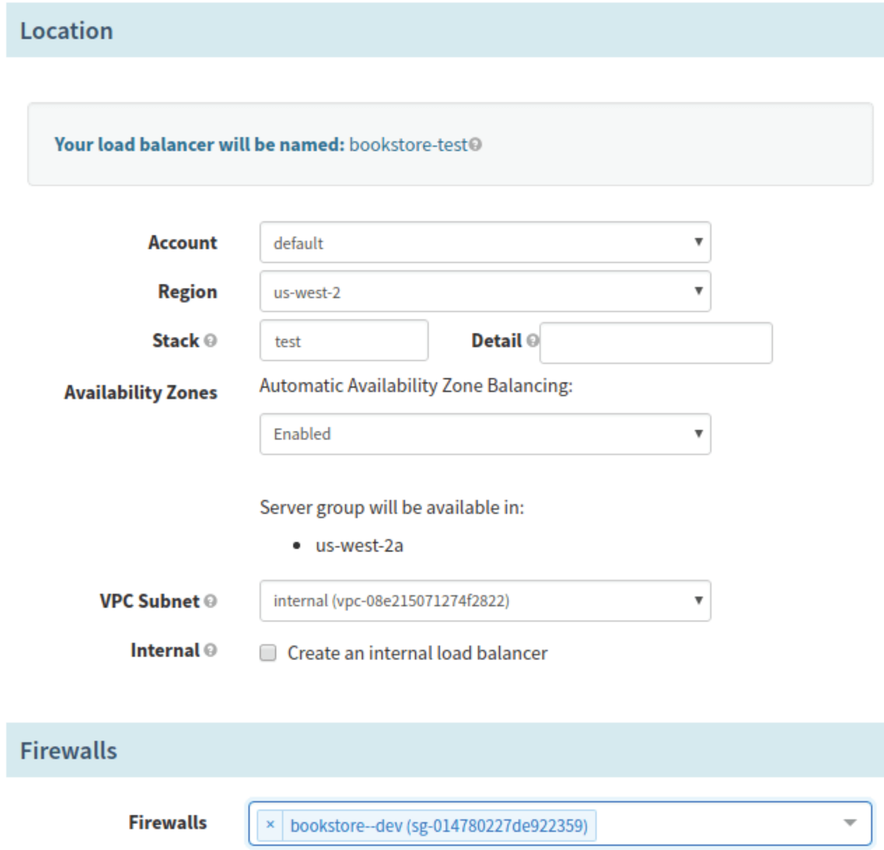
Adding a Firewall

1. Navigate to the bookstore application, **INFRASTRUCTURE > FIREWALLS**, and click **Create Firewall**.
2. Input dev in the **Detail** field.
3. Input Bookstore dev environment in the **Description** field.
4. Within the **VPC** dropdown select SpinnakerVPC.
5. Under the **Ingress** header click **Add new Firewall Rule**. Set the following **Firewall Rule** settings.
   - **Firewall:** default
   - **Protocol:** TCP
   - **Start Port:** 80
   - **End Port:** 80
6. Click the **Create** button to finalize the firewall settings.



Adding a Load Balancer

1. Navigate to the bookstore application, **INFRASTRUCTURE > LOAD BALANCERS**, and click **Create Load Balancer**.
2. Select **Classic (Legacy)** and click **Configure Load Balancer**.
3. Input test in the **Stack** field.
4. In the **VPC Subnet** dropdown select internal (vpc-...).
5. In the **Firewalls** dropdown select bookstore--dev (...).
6. Click **Create** to generate the load balancer.



Add Load Balancer Spinnaker Dialog

Creating a Pipeline in Spinnaker

The final step is to add a **pipeline**, which is where we tell Spinnaker what it should actually "do"! In this case we'll tell it to **bake** a virtual machine image containing Redis, then to **deploy** that image to our waiting EC2 instance.

1. Navigate to the `bookstore` application, **PIPELINES** and click **Create Pipeline**.
2. Select `Pipeline` in the **Type** dropdown.
3. Input `Bookstore Dev to Test` in the **Pipeline Name** field.
4. Click **Create**.

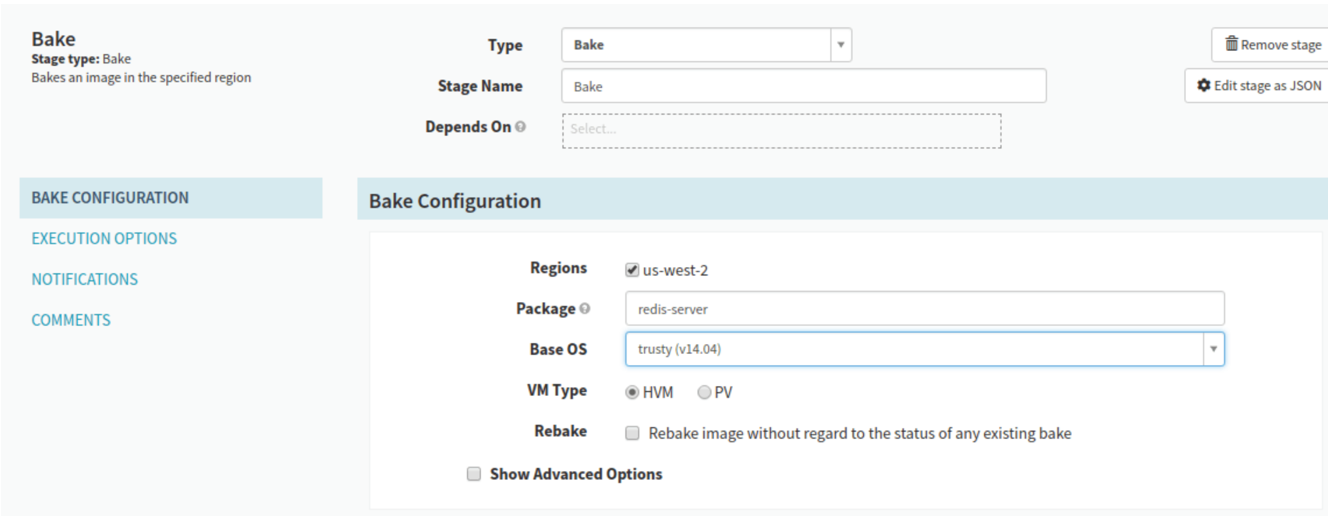
Adding a Bake Stage

1. Click the **Add stage** button.
2. Under **Type** select `Bake`.

3. Input redis-server in the **Package** field.

4. Select trusty (v14.04) in the **Base OS** dropdown.

5. Click **Save Changes** to finalize the stage.



Add Bake Deployment Stage Spinnaker Dialog

> ### Ignoring Jenkins/Travis
> In production environments you'll likely also want to incorporate Travis, Jenkins, or another CI solution as a preceding stage to the **bake** stage. Otherwise, Spinnaker will default to baking and deploying the most recently built package. For our purposes here we don't care, since we're using an unchanging image.

Adding a Deploy Stage

1. Click the **Add stage** button.

2. Under **Type** select Deploy.

3. Click the **Add server group** button to begin creating a new server group.

Adding a Server Group

1. Select internal (vpc-...) in the **VPC Subnet** dropdown.

2. Input dev in the **Stack** field.

3. Under **Load Balancers > Classic Load Balancers** select the bookstore-dev load balancer we created.

4. Under **Firewalls > Firewalls** select the bookstore--dev firewall we also created.

5. Under **Instance Type** select the **Custom Type** of instance you think you'll need. For this example we'll go with something small and cheap, such as t2.large.

6. Input 3 in the **Capacity > Number of Instances** field.

7. Under **Advanced Settings > Key Name** select the key pair name you used when deploying your Spinnaker CloudFormation stack.

8. In the A**dvanced Settings > IAM Instance Profile** field input the **Instance Profile ARN** value of the BaseIAMRole found in the **AWS > IAM > Roles > BaseIAMRole** dialog (e.g. arn:aws:iam::0 123456789012:instance-profile/BaseIAMRole).

9. We also need to ensure the user/Spinnaker-SpinnakerUser that was generated has permissions to perform to pass the role/ BasIAMRole **role** during deployment.

   1. Navigate to **AWS > IAM > Users > Spinnaker-SpinnakerUser-### > Permissions.**

   2. Expand Spinnakerpassrole policy and click **Edit Policy.**

   3. Select the **JSON** tab and you'll see the auto-generated Spinnaker-BaseIAMRole listed in Resources.

   4. Convert the Resource key value to an array so you can add a second value. Insert the **ARN** for the role/BaseIAMRole of your account (the account number will match the number above).

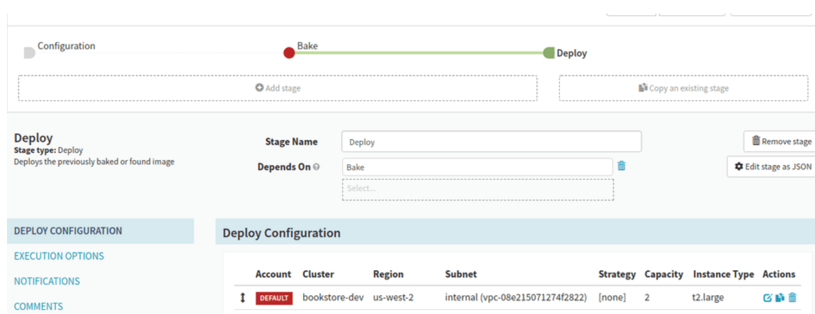
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/Spinnaker-BaseIAMRole-6D9LJ9HS4PZ7",
                "arn:aws:iam::123456789012:role/BaseIAMRole"
            ]
        }
    ]
}
```

5. Click **Review Policy** and **Save Changes**.

10. Click the **Add** button to create the deployment cluster configuration.

11. Finally, click **Save Changes** again at the bottom of the **Pipelines** interface to save the full Configuration > Bake > Deploy pipeline.

12. You should now have a Bookstore Dev to Test two-stage **pipeline** ready to go!



Executing the Pipeline

1. Navigate to the bookstore application, select **Pipelines**, and click **Start Manual Execution** next to the Bookstore Dev to Test pipeline.

2. Click **Run** to begin manual execution.

3

After waiting a few moments, assuming none of the potential setbacks below bite you, you'll shortly see output indicating the bookstore-dev-v000 **Server Group** has been successfully created. Browse over to **AWS > EC2** and you'll see your three new instances launched!

To resize this **Server Group** use the **Resize Server Group** dialog in Spinnaker. Alternatively, you can find additional options under **Server Group Actions**, such as **Disable** or **Destroy** to stop or terminate instances, respectively.

Troubleshooting Pipeline Executions

Error: Unknown configuration key ena_support

If you get an ena_support error during deployment (see: #2237) the solution is to *remove* the ena_support reference line within the builders block in the /opt/rosco/config/packer/aws-ebs.json Rosco configuration file.

```
sudo nano /opt/rosco/config/packer/aws-ebs.json
```

```
{
  "builders": {
    "aws_ena_support": "{% raw %}{{user `aws_ena_support`}}{% endraw %}",
  },
}
```

Error: 0.000000 is an invalid spot instance price

If you get such an error during deployment (see: #2889) the solution is to remove spot_price reference lines within the builders block in the /opt/rosco/config/packer/aws-ebs.json Rosco configuration file.

```
sudo nano /opt/rosco/config/packer/aws-ebs.json
```

```
{
  "builders": {
    "spot_price": "{% raw %}{{user `aws_spot_price`}}{% endraw %}",
    "spot_price_auto_product": "{% raw %}{{user `aws_spot_price_auto_product`}}{% endraw
%}",
  },
}
```

Error: Bake stage failure after provisioning install_packages.sh script

This error is typically due to an outdated install_packages.sh script. To resolve this override with the latest downloaded version.

```
sudo nano https://raw.githubusercontent.com/spinnaker/rosco/master/rosco-web/config/packer/
install_packages.sh --output /opt/rosco/config/packer/install_packages.sh
```

How to Install Chaos Monkey

Before you can use Chaos Monkey you'll need to have Spinnaker deployed and running. We've created a handful of step-by-step tutorials for deploying Spinnaker, depending on the environment and level of control you're looking for.

How to Quickly Deploy Spinnaker for Chaos Monkey will guide you through a rapid deployment of Spinnaker on AWS.

How to Deploy a Spinnaker Stack for Chaos Monkey provides a much more in-depth tutorial for installing Spinnaker as it was intended, with the help of the Halyard tool, on a local or virtual machine.

Installing MySQL

Chaos Monkey requires MySQL, so make sure it's installed on your local system.

> ### Warning
> Chaos Monkey is currently incompatible with MySQL version 8.0 or higher, so 5.X is recommended.

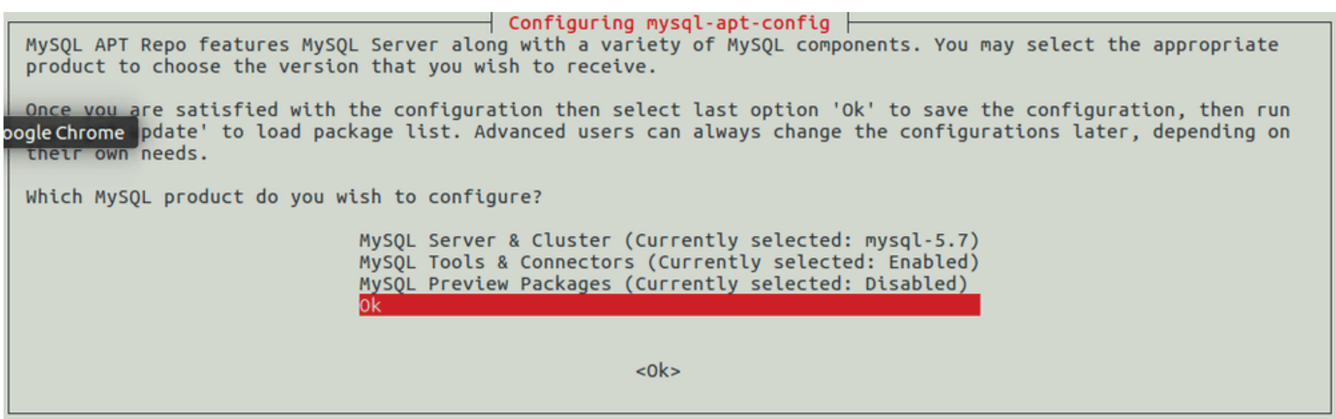
1. Download the latest `mysql-apt.deb` file from the [official website](), which we'll use to install MySQL

```
curl -OL https://dev.mysql.com/get/mysql-apt-config_0.8.10-1_all.deb
```

2. Install `mysql-server` by using the `dpkg` command.

```
sudo dpkg -i mysql-apt-config_0.8.10-1_all.deb
```

3. In the UI that appears press enter to change the **MySQL Server & Cluster** version to `mysql-5.7`. Leave the other options as default and move down to `Ok` and press `Enter` to finalize your choice.



4. Now use `sudo apt-get update` to update the MySQL packages related to the version we selected (`mysql-5.7`, in this case).

```
sudo apt-get update
```

5. Install mysql-server from the packages we just retrieved. You'll be prompted to enter a root password.

```
sudo apt-get install mysql-server
```

6. You're all set. Check that MySQL server is running with systemctl.

```
systemctl status mysql
```

7. (Optional) You may also wish to issue the mysql_secure_installation command, which will walk you through a few security-related prompts. Typically, the defaults are just fine.

Setup MySQL for Chaos Monkey

1. Launch the mysql CLI as the root user.

```
mysql -u root -p
```

2. Create a chaosmonkey database for Chaos Monkey to use.

```
CREATE DATABASE chaosmonkey;
```

3. Add a chaosmonkey MySQL user.

```
CREATE USER 'chaosmonkey'@'localhost' IDENTIFIED BY
'password';
```

4. Grant all privileges in the chaosmonkey database to the new chaosmonkey user.

```
GRANT ALL PRIVILEGES ON chaosmonkey.* TO
'chaosmonkey'@'localhost';
```

5. Finally, save all changes made to the system.

```
FLUSH PRIVILEGES;
```

Installing Chaos Monkey

1. Optional) Install go if you don't have it on your local machine already.
   1. Go to this download page and download the latest binary appropriate to your environment.

```
curl -O https://dl.google.com/go/go1.11.linux-amd64.tar.gz
```

2. Extract the archive to the /usr/local directory.

```
sudo tar -C /usr/local -xzf go1.11.linux-amd64.tar.gz
```

3. Add /usr/local/go/bin to your $PATH environment variable.

```
export PATH=$PATH:/usr/local/go/bin
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
```

2. (Optional) Check if the $GOPATH and $GOBIN variables are set with echo $GOPATH and echo $GOBIN. If not, export them and add them to your bash profile.

```
export GOPATH=$HOME/go
echo 'export GOPATH=$HOME/go' >> ~/.bashrc
export GOBIN=$HOME/go/bin
echo 'export GOBIN=$HOME/go/bin' >> ~/.bashrc
export PATH=$PATH:$GOBIN
echo 'export PATH=$PATH:$GOBIN' >> ~/.bashrc
```

3. Install the latest Chaos Monkey binary.

Configure Spinnaker for Chaos Monkey

Spinnaker includes the Chaos Monkey feature as an option, but it is disabled by default.

1. (Optional) If necessary, enable the Chaos Monkey feature in your Spinnaker deployment.

   - On a Halyard-based Spinnaker deployment you must enable the Chaos Monkey feature via the Halyard --chaos flag.

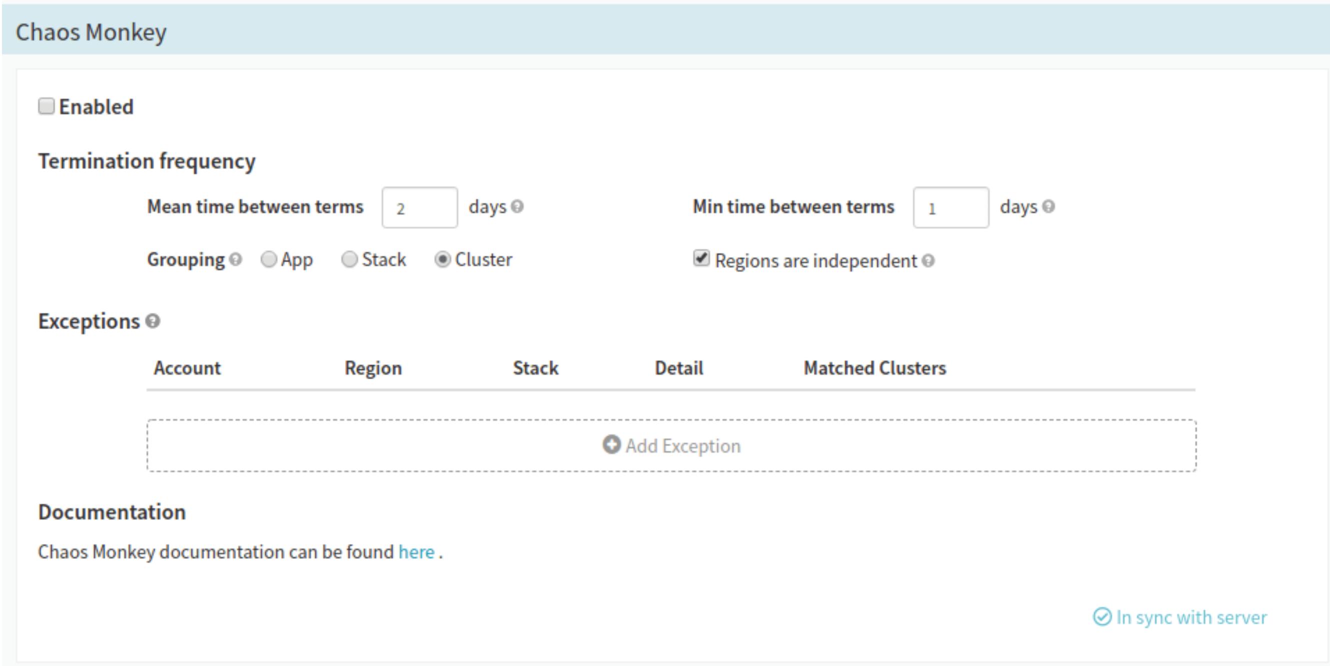
   ```
   hal config features edit --chaos true
   ```

   - On a quick start Spinnaker deployment you'll need to manually enable the Chaos Monkey feature flag within the /opt/deck/html/settings.js file. Make sure the var chaosEnabled is set to true, then save and reload Spinnaker.

   ```
   sudo nano /opt/deck/html/settings.js
   ```

   ```
   // var chaosEnabled = ${services.chaos.enabled};
   var chaosEnabled = true;
   ```

2. Navigate to **Applications > (APPLICATION_NAME) > CONFIG** and select **CHAOS MONKEY** in the side navigation.



Chaos Monkey Spinnaker Dialog

3. Check the **Enabled** box to enable Chaos Monkey.

4. The UI provides useful information for what every option does, but the most important options are the **mean** and **min** times between instance termination. If your setup includes multiple clusters or stacks, altering the **grouping** may also make sense. Finally, you can add **exceptions** as necessary, which acts as a kind of whitelist of instances that will be ignored by Chaos Monkey, so you can keep the most critical services up and running.

5. Once your changes are made click the **Save Changes** button.

How to Configure Chaos Monkey

1. Start by creating the chaosmonkey.toml, which Chaos Monkey will try to find in all of the following locations, until a configuration file is found:
   - (current directory)
   - /apps/chaosmonkey
   - /etc
   - /etc/chaosmonkey

> Generally, if you're configuring multiple Chaos Monkey installations on the same machine you should use application-specific configurations, so putting them in separate directories is ideal. However, if you're just using one installation on the machine then /apps/chaosmonkey/chaosmonkey.toml works well.

2. Add the following basic configuration structure to your chaosmonkey.toml file, replacing appropriate <DATABASE_> configuration values with your own settings.

```
[chaosmonkey]
enabled = true
schedule_enabled = true
leashed = false
accounts = ["aws-primary"]

start_hour = 9      # time during day when starts terminating
end_hour = 15       # time during day when stops terminating

# location of command Chaos Monkey uses for doing terminations
term_path = "/apps/chaosmonkey/chaosmonkey-terminate.sh"

# cron file that Chaos Monkey writes to each day for scheduling kills
cron_path = "/etc/cron.d/chaosmonkey-schedule"

[database]
host = "localhost"
name = "<DATABASE_NAME>"
user = "<DATABASE_USER>"
encrypted_password = "<DATABASE_USER_PASSWORD>"

[spinnaker]
endpoint = "http://localhost:8084"
```

3. With Chaos Monkey configured it's time to migrate it to the MySQL

```
$ chaosmonkey migrate
[16264] 2018/09/04 14:11:16 Successfully applied database migrations. Number of migrations applied:  1
[16264] 2018/09/04 14:11:16 database migration applied successfully
```

Error: 1298: Unknown or incorrect time zone: 'UTC'

If you experience a timezone error this typically indicates a configuration problem with MySQL. Just run the `mysql_tzinfo_to_sql` command to update your MySQL installation.

```
mysql_tzinfo_to_sql /usr/share/zoneinfo/ | mysql -u root
```

How to Use Chaos Monkey

Using the `chaosmonkey` command line tool is fairly simple. Start by making sure it can connect to your `spinnaker` instance with `chaosmonkey config spinnaker`.

```
chaosmonkey config spinnaker
```

```
# OUTPUT
(*chaosmonkey.AppConfig)(0xc00006ca00)({
 Enabled: (bool) true,
 RegionsAreIndependent: (bool) true,
 MeanTimeBetweenKillsInWorkDays: (int) 2,
 MinTimeBetweenKillsInWorkDays: (int) 1,
 Grouping: (chaosmonkey.Group) cluster,
 Exceptions: ([]chaosmonkey.Exception) {
 },
 Whitelist: (*[]chaosmonkey.Exception)(<nil>)
})
```

Error: 1298: Unknown or incorrect time zone: 'UTC'

If you're running Spinnaker on Kubernetes you can use the `kubectl get nodes --watch` |ommand to keep track of your Kubernetes nodes while running Chaos Experiments.

```
kubectl get nodes --watch
```

```
# OUTPUT
ip-10-100-11-239.us-west-2.compute.internal  Ready    <none>   3d     v1.10.3
ip-10-100-10-178.us-west-2.compute.internal  Ready    <none>   3d     v1.10.3
ip-10-100-10-210.us-west-2.compute.internal  Ready    <none>   3d     v1.10.3
```

To manually terminate an instance with Chaos Monkey use the `chaosmonkey terminate` command.

```
chaosmonkey terminate <app> <account>
[--region=<region>] [--stack=<stack>] [--cluster=<cluster>]
[--leashed]
```

For this example our **application** is `spinnaker` and our **account** is `aws-primary`, so using just those two values and leaving the rest default should work.

```
chaosmonkey terminate spinnaker aws-primary
```

```
# OUTPUT
[11533] 2018/09/08 18:39:26 Picked: {spinnaker aws-
primary us-west-2 eks spinnaker-eks-nodes-NodeGroup-
KLBYTZDP0F89 spinnaker-eks-nodes-NodeGroup-
KLBYTZDP0F89 i-054152fc4ed41d7b7 aws}
```

Now look at the AWS EC2 console (or at the terminal window running `kubectl get nodes --watch`) and after a moment you'll see one of the instances has been terminated.

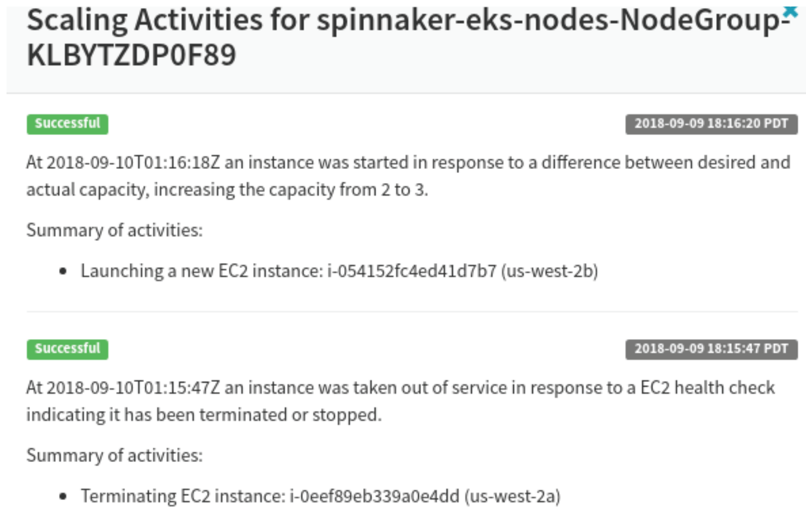
```
ip-10-100-10-178.us-west-2.compute.internal  Ready     <none>   3d     v1.10.3
ip-10-100-11-239.us-west-2.compute.internal  Ready     <none>   3d     v1.10.3
ip-10-100-10-210.us-west-2.compute.internal  NotReady  <none>   3d     v1.10.3
ip-10-100-10-178.us-west-2.compute.internal  Ready     <none>   3d     v1.10.3
ip-10-100-11-239.us-west-2.compute.internal  Ready     <none>   3d     v1.10.3
```

If you quickly open up the Spinnaker Deck web interface you'll see only two of the three instances in the cluster are active, as we see in `kubectl` above. However, wait a few more moments and Spinnaker will notice the loss of an instance, recognize it has been stopped/terminated due to an EC2 health check, and will immediately propagate a new instance to replace it, thus ensuring the server group's desired capacity remains at `3` instances.

For Kubernetes Spinnaker deployments, a `kubectl get nodes --watch` output confirms these changes (in this case, the new local `ip-10-100-11-180.us-west-2.compute.internal` instance was added).

```
ip-10-100-11-239.us-west-2.compute.internal   Ready      <none>   3d    v1.10.3
ip-10-100-10-178.us-west-2.compute.internal   Ready      <none>   3d    v1.10.3
ip-10-100-11-180.us-west-2.compute.internal   NotReady   <none>   10s   v1.10.3
ip-10-100-11-239.us-west-2.compute.internal   Ready      <none>   3d    v1.10.3
ip-10-100-10-178.us-west-2.compute.internal   Ready      <none>   3d    v1.10.3
ip-10-100-11-180.us-west-2.compute.internal   Ready      <none>   20s   v1.10.3
```

Spinnaker also tracks this information. Navigating to the your Spinnaker application **INFRASTRUCTURE > CLUSTERS >** `spinnaker-eks-nodes-NodeGroup` **> CAPACITY** and click **View Scaling Activities** to see the Spinnaker scaling activities log for this node group. In this case we see the successful activities that lead to the health check failure and new instance start.



How to Schedule Chaos Monkey Terminations

Before we get to scheduling anything you'll want to copy the `chaosmonkey` executable to the `/apps/chaosmonkey` directory. While you can leave it in the default `$GOBIN` directory, it'll be easier to use with cron jobs and other system commands if it's in a global location.

```
sudo cp ~/go/bin/chaosmonkey /apps/chaosmonkey/
```

Now that we've confirmed we can manually terminate instances via Chaos Monkey you may want to setup an automated system for doing so. The primary way to do this is to create a series of scripts that regenerate a unique `crontab` job that is scheduled to execute on a specific date and time. This cron job is created every day (or however often you like), and the execution time is randomized based on the `start_hour`, `end_hour`, and `time_zone` settings in the `chaosmonkey.toml` configuration. We'll be using four files for this: Two crontab files and two bash scripts.

1. Start by creating the four files we'll be using for this.

```
sudo touch /apps/chaosmonkey/chaosmonkey-schedule.sh
sudo touch /apps/chaosmonkey/chaosmonkey-terminate.sh
sudo touch /etc/cron.d/chaosmonkey-schedule
sudo touch /etc/cron.d/chaosmonkey-daily-scheduler
```

2. Now set executable permissions for the two bash scripts so the cron (root) user can execute them.

```
sudo chmod a+rx /apps/chaosmonkey/chaosmonkey-schedule.sh
sudo chmod a+rx /apps/chaosmonkey/chaosmonkey-terminate.sh
```

3. Now we'll add some commands to each script in the order they're expected to call one another. First, the `/etc/cron.d/chaosmonkey-daily-scheduler` is executed once a day at a time you specify. This will call the `/apps/chaosmonkey/chaosmonkey-schedule.sh` script, which will perform the actual scheduling for termination. Paste the following into `/etc/cron.d/chaosmonkey-daily-scheduler` (as with any cron job you can freely edit the schedule to determine when the cron job should be executed).

```
# min  hour  dom  month  day  user  command
0    12   *   *    *   root  /apps/chaosmonkey/chaosmonkey-schedule.sh
```

4. The `/apps/chaosmonkey/chaosmonkey-schedule.sh` script should perform the actual `chaosmonkey schedule` command, so paste the following into `/apps/chaosmonkey/chaosmonkey-schedule.sh`.

```
#!/bin/bash
/apps/chaosmonkey/chaosmonkey schedule >> /var/log/chaosmonkey-schedule.log 2>&1
```

5. When the `chaosmonkey schedule` command is called by the `/apps/chaosmonkey/chaosmonkey-schedule.sh` script it will automatically write to the `/etc/cron.d/chaosmonkey-schedule` file with a randomized timestamp for execution based on the Chaos Monkey configuration. Here's an example of what the generated `/etc/cron.d/chaosmonkey-schedule` looks like.

```
# /etc/cron.d/chaosmonkey-schedule
9 16 9 9 0 root /apps/chaosmonkey/chaosmonkey-terminate.sh spinnaker aws-primary --cluster=spinnaker-eks-nodes-NodeGroup-KLBYTZDP0F89 --region=us-west-2
```

6. Lastly, the `/apps/chaosmonkey/chaosmonkey-terminate.sh` script that is called by the generated `/etc/cron.d/chaosmonkey-schedule` cron job should issue the chaosmonkey terminate command and output the result to the log. Paste the following into `/apps/chaosmonkey/chaosmonkey-terminate.sh`.

```
#!/bin/bash
/apps/chaosmonkey/chaosmonkey terminate "$@" >> /var/log/chaosmonkey-terminate.log 2>&1
```

Next Steps

You're all set now! You should have a functional Spinnaker deployment with Chaos Monkey enabled, which will perform a cron job once a day to terminate random instances based on your configuration!

Chaos Monkey is just the tip of the Chaos Engineering iceberg, and there are a lot more failure modes you can experiment with to learn about your system.

The rest of this guide will cover the other tools in The Simian Army family, along with an in-depth look at the Chaos Monkey Alternatives.

We built Gremlin to provide a production-ready framework to safely, securely, and easily simulate real outages with an ever-growing library of attacks.

ADVANCED DEVELOPER GUIDE

Taking Chaos Monkey to the Next Level

This chapter provides advanced developer tips for Chaos Monkey and other Chaos Engineering tools, including tutorials for manually deploying Spinnaker stacks on a local machine, virtual machine, or with Kubernetes. From there you can configure and deploy Spinnaker itself, along with Chaos Monkey and other Chaos Engineering tools!

How to Install AWS CLI

Start by installing the AWS CLI tool on your machine, if necessary. doing so is to use the CloudFormation Quick Start template.

Simplifying AWS Credentials

You can make future AWS CLI commands easier by creating AWS `profiles`, which will add | configuration and credentials to the local `~/.aws/credentials` file. In some cases you'll | be using two different accounts/profiles, so you can add the credentials for multiple | accounts to `~/.aws/credentials` by using `aws configure --profile <profile-name>` commands.

```
aws configure --profile spinnaker-developer
AWS Access Key ID [None]: <AWS_ACCESS_KEY_ID>
AWS Secret Access Key [None]: <AWS_SECRET_
ACCESS_KEY>
Default region name [None]: us-west-2
Default output format [None]: text

aws configure --profile primary
AWS Access Key ID [None]: <AWS_ACCESS_KEY_ID>
AWS Secret Access Key [None]: <AWS_SECRET_
ACCESS_KEY>
Default region name [None]: us-west-2
Default output format [None]: text
```

In the future, simply add the `--profile <profile-name>` flag to any AWS CLI command to | force AWS CLI to use that account.

How to Install Halyard

Halyard is the CLI tool that manages Spinnaker deployments and is typically the first step to any manual Spinnaker setup.

1. Download Halyard installation script
   - For Debian/Ubuntu.

```
curl -O https://raw.githubusercontent.com/
spinnaker/halyard/master/install/debian/
InstallHalyard.sh
```

   - For MacOS.

```
curl -O https://raw.githubusercontent.com/
spinnaker/halyard/master/install/macos/
InstallHalyard.sh
```

2. Install Halyard with the InstallHalyard.sh script. If prompted, default options are usually just fine.

```
sudo bash InstallHalyard.sh
```

3. Source your recently-modified `.bashrc` file (or `~/.bash_profile`).

```
. ~/.bashrc
```

4. Verify Halyard was installed by checking the version.

```
hal -v
# OUTPUT
1.9.1-20180830145737
```

5. That's the basics! Halyard is now ready to be configured and used for manual or quick start Spinnaker deployments.

How to Install Spinnaker

This section walks you through the most basic Spinnaker installation process, suitable for simple Spinnaker deployments.

1. Use the `hal version list` command to view the current Spinnaker version list.

```
hal version list
```

2. Configure Halyard to use the latest version of Spinnaker.

```
hal config version edit --version 1.9.2
```

3. *(Optional)* Enable Chaos Monkey in the Halyard config.

```
hal config features edit --chaos true
```

4. Tell Halyard what type of environment you're deploying Spinnaker to. Most production setups will want to use Kubernetes or another distributed solution, but the default deployment is a local installation. The `hal config deploy edit --type` flag can be used to change the environment.

```
hal config deploy edit --type localdebian
```

5. Halyard requires some form of persistent storage, so we'll use AWS S3 for simplicity. Modify the Halyard config and be sure to pass an AWS `ACCESS_KEY_ID` and `SECRET_ACCESS_KEY` with privileges to create and use S3 buckets.

```
hal config storage s3 edit --access-key-id <AWS_ACCESS_
KEY_ID> --secret-access-key --region us-west-2
```

6. Configure Halyard to use the `s3` storage type.

```
hal config storage edit --type s3
```

7. Now use `sudo hal deploy apply` to deploy Spinnaker to the local machine.

```
sudo hal deploy apply
```

8. After deployment finishes you should have a functioning Spinnaker installation! If you've installed on your local machine you can navigate to the Spinnaker Deck UI at localhost:9000 to see it in action. If Spinnaker was deployed on a remote machine use the `hal deploy connect` command to quickly establish SSH tunnels and connect.

Next Steps

You're now ready to install and then start using Chaos Monkey or other Simian Army tools.

How to Deploy a Spinnaker Stack for Chaos Monkey

Manually deploying Spinnaker with the help of Halyard is the best way to have the utmost control over your Spinnaker installation, and is ideal for advanced deployments to EC2 instances, EKS/Kubernetes clusters, and the like. Choose one of the three options depending on your needs.

Deploying a Spinnaker Stack with AWS Console

Deploying a Spinnaker Stack with AWS CLI

Deploying a Spinnaker Stack with Kubernetes

After Spinnaker is running on your chosen platform proceed to our How to Install Chaos Monkey guide to get started with Chaos Monkey!

Deploying a Spinnaker Stack with AWS Console

This section will guide you through deploying a Spinnaker stack with the AWS web console.

Prerequisites

Install Halyard

Install AWS CLI

1. In AWS navigate to **CloudFormation** and click **Create Stack**.
2. Download this `managing.yaml` file to your local machine.
3. Under **Choose a template** click the **Choose File** button under **Upload a template to Amazon S3** and select the downloaded `managing.yaml`.
4. Click **Next**.
5. Input `spinnaker-managing-infrastructure-setup` into the **Stack name** field.
6. Select `false` in the **UseAccessKeyForAuthentication** dropdown.
7. Click **Next**.
8. On the **Options** screen leave defaults and click **Next**
9. Check the **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. checkbox and click **Create** to generate the stack.

> **Note**
>
> If your AWS account already contains the BaseIAMRole AWS::IAM::ROLE you may have to delete it before this template will succeed.

10. Once the spinnaker-managing-infrastructure-setup stack has a CREATE_COMPLETE **Status**, select the **Outputs** tab and copy all key/value pairs there somewhere convenient. They'll look something like the following.

Key	Value
VpcId	vpc-0eff1ddd5f7b26ffc
ManagingAccountId	123456789012
AuthArn	arn:aws:iam::123456789012:role/ SpinnakerAuthRole
SpinnakerInstanceProfileArn	arn:aws:iam::123456789012 :instance-profile/spinnaker-managing-infrastructure-setup-SpinnakerInstanceProfile-1M72QQCCLNCZ9
SubnetIds	subnet-0c5fb1e7ab00f20a7,subnet-065af1a1830937f86

11. Add a new AWS account to Halyard named spinnaker-developer with your AWS account id and your appropriate AWS region name.

```
hal config provider aws account add spinnaker-developer \
--account-id 123456789012 \
--assume-role role/spinnakerManaged \
--regions us-west-2
```

12. Enable AWS in Halyard.

```
hal config provider aws enable
```

Next Steps

You're now ready to install Spinnaker and install Chaos Monkey to begin Chaos Experimentation!

Deploying a Spinnaker Stack with AWS CLI

This section will guide you through deploying a Spinnaker stack with the AWS CLI tool.

Prerequisites

Install Halyard

Install AWS CLI

1. Download this managing.yaml template.

```
curl -OL https://d3079gxvs8ayeg.cloudfront.net/templates/managing.yaml
```

2. Now we'll use AWS CLI to create the spinnaker-managing-infrastructure stack via CloudFormation. We want to use the primary or managing account for this, so we'll specify the --profile primary, which will grab the appropriate credentials, region, and so forth.

```
aws cloudformation deploy --stack-name
spinnaker-managing-infrastructure --template-
file managing.yaml --parameter-overrides
UseAccessKeyForAuthentication=true --capabilities
CAPABILITY_NAMED_IAM --profile primary
```

Error: Unresolved resource dependency for SpinnakerInstanceProfile.

If you receive the above error while creating the spinnaker-managing-infrastructure stack you may need to edit the managing.yaml file and comment out the two SpinnakerInstanceProfileArn related lines under the Outputs block.

```
# ...
Outputs:
# ...
#  SpinnakerInstanceProfileArn:
#    Value: !GetAtt SpinnakerInstanceProfile.Arn
```

1. Once the spinnaker-managing-infrastructure stack has been created open the AWS console, navigate to the CloudFormation service, select the **Outputs** tab of the spinnaker-managing-infrastructure stack. We'll be using the ManagingAccountId and AuthArn values in the next step, which look something like the following.

Key	Value
ManagingAccountId	123456789012
AuthArn	arn:aws:iam::123456789012:user/spinnaker-managing-infrastructure-SpinnakerUser-15UU17KIS3EK1

2. Download this managed.yaml template.

```
curl -OL https://d3079gxvs8ayeg.cloudfront.net/templates/
managed.yaml
```

3. Now enter the following command to create the companion spinnaker-managed-infrastructure stack in CloudFormation. Be sure to specify the profile value and paste the appropriate ManagingAccountId and AuthArn values from above.

```
curl -OL https://d3079gxvs8ayeg.cloudfront.net/templates/
managed.yaml
```

4. Add your AWS Access Id and Secret Keys to Halyard.

```
hal config provider aws edit --access-key-id <AWS_
ACCESS_KEY_ID> --secret-access-key
```

5. Add your default managing account to Spinnaker.

```
hal config provider aws account add default --account-
id 123456789012 --assume-role role/spinnakerManaged
--regions us-west-2
```

6. Enable the AWS provider.

```
hal config provider aws enable
```

Spinnaker uses accounts added via the Halyard hal config provider aws account API to handle all actions performed within the specified provider (such as AWS, in this case). For this example we'll just be using our primary managing account, but you can freely add more accounts as needed.

Next Steps

With Spinnaker configured it's now time to install Spinnaker and then install Chaos Monkey.

Deploying a Spinnaker Stack with Kubernetes

Follow these steps to setup a CloudFormation EKS/Kubernetes stack for Spinnaker and Chaos Monkey.

Prerequisites

Install Halyard

Install AWS CLI

Stack Configuration

If you need to configure the stack to your own particular needs you can easily edit the template YAML as necessary. For example, in this guide we're only using a single **managing** account to handle Spinnaker/Kubernetes in AWS, but if you need to also include additional **managed** accounts you'll want to add their respective AWS ARN strings to the managing.yaml file around this line.

1. Download this managing.yaml template.

```
curl -O https://d3079gxvs8ayeg.cloudfront.net/templates/managing.yaml
```

2. Now we'll use AWS CLI to issue a cloudformation deploy command to create a new spinnaker-managing-infrastructure-setup stack using the managing.yaml template. From here on out this guide will use explicit names where applicable, but feel free to customize options as you see fit (such as the **stack name, EksClusterName**, and so forth).

```
aws cloudformation deploy --stack-name spinnaker-managing-infrastructure-setup --template-file managing.yaml --capabilities CAPABILITY_NAMED_IAM \

--parameter-overrides
UseAccessKeyForAuthentication=false
EksClusterName=spinnaker-cluster
```

3. The step above takes 10 - 15 minutes to complete, but once complete issue the following commands, which will use the AWS CLI to assign some environment variables values from the spinnaker-managing-infrastructure-setup stack we just created. We'll be using these values throughout the remainder of this guide.

```
VPC_ID=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`VpcId`].OutputValue'
--output text)
CONTROL_PLANE_SG=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`SecurityGroups`].
OutputValue' --output text)
AUTH_ARN=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`AuthArn`].OutputValue'
--output text)
SUBNETS=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`SubnetIds`].OutputValue'
--output text)
MANAGING_ACCOUNT_ID=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`ManagingAccountId`].
OutputValue' --output text)
EKS_CLUSTER_ENDPOINT=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`EksClusterEndpoint`].
OutputValue' --output text)
EKS_CLUSTER_NAME=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`EksClusterName`].
OutputValue' --output text)
EKS_CLUSTER_CA_DATA=$(aws cloudformation describe-stacks --stack-name spinnaker-managing-
infrastructure-setup --query 'Stacks[0].Outputs[?OutputKey==`EksClusterCA`].OutputValue'
--output text)
SPINNAKER_INSTANCE_PROFILE_ARN=$(aws cloudformation describe-stacks
--stack-name spinnaker-managing-infrastructure-setup --query 'Stacks[0].
Outputs[?OutputKey==`SpinnakerInstanceProfileArn`].OutputValue' --output text)
```

> You can easily output the value of an exported variable with `echo $VARIABLE_NAME`. However, remember that unless you `export` these values they only temporarily exist in the console in which you issued the commands. You may need to reissue the above commands later in the guide if you change terminal windows, so keep them handy.

4. Download this `managed.yaml` template. This template will create the `spinnakerManaged` **AWS::IAM::Role** that Spinnaker can use.

```
curl -O https://d3079gxvs8ayeg.cloudfront.net/templates/
managed.yaml
```

5. Execute this secondary CloudFormation deployment using the `managed.yaml`. Notice that this command (and many following commands) use some of the environmental variables we assigned previously, so the first stack deployment will need to be complete first.

```
aws cloudformation deploy --stack-name spinnaker-managed-infrastructure-setup --template-
file managed.yaml --capabilities CAPABILITY_NAMED_IAM \
--parameter-overrides AuthArn=$AUTH_ARN ManagingAccountId=$MANAGING_ACCOUNT_ID
```

> If the second step of deploying `spinnaker-managing-infrastructure-setup` hasn't completed yet, feel free to skip this step for the time being and proceed with installing `kubectl` and `AWS IAM Authenticator` below. Just return to this step before moving past that point.

Next Steps

You now have an EKS/Kubernetes CloudFormation stack ready for Spinnaker. You can now proceed with the deployment of Spinnaker on Kubernetes, and then move on to installing and using Chaos Monkey. If Chaos Monkey doesn't suit all your Chaos Engineering needs check out our Chaos Monkey Alternatives chapter.

How to Deploy Spinnaker on Kubernetes

This guide will walk you through the entire process of setting up a Kubernetes cluster via AWS EKS, attaching some worker nodes (i.e. EC2 instances), deploying Spinnaker to manage the Kubernetes cluster, and then using Chaos Monkey and other Simian Army tools on it! If you're looking for a simpler Spinnaker installation, you might be interested in our Spinnaker AWS Quick Start guide.

Prerequisites

> **Install Halyard**
> **Install AWS CLI**
> **Deploy a Spinnaker Stack for Kubernetes**

Install Kubectl

We'll need to install the `kubectl` client and the `AWS IAM Authenticator for Kubernetes`, which will allow Amazon EKS to use IAM for authentication to our Kubernetes cluster.

1. Download the appropriate `kubectl` binary. *Note: For the remainder of this guide we'll be using Linux examples, but most everything applies to other environments.*

- Linux: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/kubectl
- MacOS: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/darwin/amd64/kubectl
- Windows: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/windows/amd64/kubectl.exe

```
curl -O https://amazon-eks.s3-us-west-2.amazonaws.
com/1.10.3/2018-07-26/bin/linux/amd64/kubectl
```

2. Change permissions of `kubectl` so it's executable.

```
chmod +x ./kubectl
```

3. Move `kubectl` to an appropriate `bin` directory and add to your `PATH`, if necessary.

```
cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/
bin:$PATH
```

4. Verify that `kubectl` is installed. The `--client` flag is used here since we're just checking for the local installation, not making any connections yet.

```
kubectl version --client
```

Install AWS IAM Authenticator

We'll follow the same basic steps as above to install the AWS IAM Authenticator as well.

1. Download the appropriate `aws-iam-authenticator` binary.
   - Linux: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/aws-iam-authenticator
   - MacOS: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/darwin/amd64/aws-iam-authenticator

- Windows: https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/windows/amd64/aws-iam-authenticator.exe

```
curl -O https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/aws-iam-authenticator
```

2. Make `aws-iam-authenticator` executable.

```
chmod +x ./aws-iam-authenticator
```

3. Move `aws352-iam-authenticator` to an appropriate `bin` directory and add to your `PATH`, if necessary.

```
cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$HOME/bin:$PATH
```

4. Test the `aws-iam-authenticator` installation.

```
aws-iam-authenticator help
```

Configure Kubectl

Prerequisites

> Make sure you've gone through the Deploying a Spinnaker Stack with Kubernetes section.

With everything setup you can now edit the `kubectl` configuration files, which will inform `kubectl` how to connect to your Kubernetes/EKS cluster.

1. Copy and paste the following into the `~/.kube/config` file.

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "<cluster-name>"
        # - "-r"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

Configuring Multiple Kubernetes Clusters

This guide assumes you're just configuring `kubectl` to handle a single Kubernetes cluster, but if you need to configure and handle multiple clusters the convention for doing so is to create a unique `config` file for each cluster. Simply name each `config` file `~/.kube/config-<cluster-name>`, where `<cluster-name>` is replaced by the name of the Kubernetes cluster you already created.

Using a Specific `AWS::IAM::Role`

If you need to have the AWS IAM Authenticator and kubectl use a specific `Role` then uncomment the `- "-r"` and `- "<role-arn>"` lines and paste the `AWS::IAM::Role` ARN in place of `<role-arn>`.

2. Replace the `<...>` placeholder strings with the following `EKS_CLUSTER_` environmental variable values from earlier:
    - `<endpoint-url>: $EKS_CLUSTER_ENDPOINT`
    - `<base64-encoded-ca-cert>: $EKS_CLUSTER_CA_DATA`
    - `<cluster-name>: $EKS_CLUSTER_NAME`
3. Your `~/.kube/config` file should now look something like this:

```
apiVersion: v1
clusters:
- cluster:
    server: https://51A93D4A4B1228D6D06BE160DA2D3A8A.yl4.us-west-2.eks.amazonaws.com
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRFN
E1Ea3dOakExTVRnME1BHeGFNN3QyaApzcm8wa1ZOWTdCQ1g5YVBBY09rRFVuRExGektpZTJnZVZLOFpxVHJvZEpLR0p3SEtjVDJNNUtsR0ZTSjMzSGpoCk5ZeVErTnJkd0VKT2puR2xZN3R1eVFvZj
hnNU0vNWZzZSt0TWFMTjJjQ3NWNFA1NCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRRUxCUUFEZ2dFQkFKN3owTEI5NFVoZWNwTUh0VGYrVTkxVDlxU2IKNWFVRGQrdlVTNEpVTWwwdk01OXBqc0
5CNDU1Z1l6ZkpLelZ1YXI5TjJOVURiRElINUJsbjlCRjWb1hEVEk0TURrd016QTFNVGcwTVZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBR
GdnRVBBRENDQVFvQ2dnRUJBT0h6CldrZ2pzTWZ0eEJYd3NZOGRuVXI5UUQTAzZXczazlaZHZlMWNYRlp4bHdqc3RSdWN3eUxRTG12eUh0VzJsTjE4RENqSXF5OGwxeUlYSENERQpXQjI3eHo4TXg3Z
DJVSjIyaThjQ0F3RUFBYU1qTUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0V5OSTJHYjV4QU1vYjJBaWQwbEQrT2NncGdDcXQvQ3h2SlFJRGpxbjRKT1AKejh6RVkvWVVsQjBUOTVXUU
FsRE9ZWnlKY3lDeWxYcFZRTnNDRWNSMFhUakRaVDFVbXMyMmk4NlozYy8xQ1IrWgpKNkNqZ3IvZkNadVVaV0VUbGt1WXhlSG5CQS91ZURJM1NsMVdnY0ZFMGFyNGxsVkVFVngyS01PZXhuM09FdHI0
CjhBd1dmQWxzSUNXRWdjMjRKdzk5MG9LelNObXB0cWRaOEFwczhVaHJoZWtoNEh1blpFLzhud1prb213SE1TcTYKbjl5NFJN3RyR0xWN0RzMUxWUFBlWjkKVVB0eU1WODlieFVEeFhNV3I3d2tNRy9
YckdtaC9nN1gwb1grdXRnUUtiSWdPaHZMZEFKSDNZUUlyTjhHS0krcwpIMGtjTnpYMWYzSGdabUVINUIxNXhER0R2SnA5a045Q29VdjRYVE5tdlljVlNVSy9vcWdwaXd1TU9oZz0KLS0tLS1FTkQgQ
0VSVElGSUNBVEUtLS0tLQo=
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "spinnaker-cluster"
        # - "-r"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

4. Save your `config` file and export the `KUBECONFIG` variable to include the new `config` location.

```
export KUBECONFIG=$KUBECONFIG:~/.kube/config
```

5. Verify that `kubectl` is able to use your credentials to connect to your cluster with `kubectl get svc`:

```
kubectl get svc

NAME       TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
kubernetes   ClusterIP   172.25.0.1   <none>
```

Create AWS Accounts and Roles

We now need to apply service accounts and roles to the `kubectl` `spinnaker` namespace. In Kubernetes, a namespace is a *virtual* cluster, which can join other virtual clusters and all be housed within the same *physical* cluster.

1. Issue the following commands to create the `spinnaker` namespace, `spinnaker-service-account` service account, and `spinnaker-admin` binding. The `apply` command in `kubectl` applies a configuration based on the passed resource (YAML files, in this case).

```
CONTEXT=aws
kubectl create namespace spinnaker
kubectl apply -f https://d3079gxvs8ayeg.cloudfront.net/
templates/spinnaker-service-account.yaml
kubectl apply -f https://d3079gxvs8ayeg.cloudfront.net/
templates/spinnaker-cluster-role-binding.yaml
```

2. Next, we're creating the authentication TOKEN environmental variable.

```
TOKEN=$(kubectl get secret --context $CONTEXT \
 $(kubectl get serviceaccount spinnaker-service-account \
   --context $CONTEXT \
   -n spinnaker \
   -o jsonpath='{.secrets[0].name}') \
 -n spinnaker \
 -o jsonpath='{.data.token}' | base64 --decode)
```

3. Pass the TOKEN to the following configuration commands to set kubectl credentials.

```
kubectl config set-credentials ${CONTEXT}-token-user
--token $TOKEN
kubectl config set-context $CONTEXT --user ${CONTEXT}-
token-user
```

Add Kubernetes Provider to Halyard

The next step is to add Kubernetes as a provider to Halyard/Spinnaker. A provider is just an interface to one of many virtual resources Spinnaker will utilize. AWS, Azure, Docker, and many more are all considered providers, and are managed by Spinnaker via accounts.

1. Start by enabling the Kubernetes provider in Halyard.4

```
hal config provider kubernetes enable
```

2. Add the kubernetes-master account to Halyard.

```
hal config provider kubernetes account add kubernetes-
master --provider-version v2 --context $(kubectl config
current-context)`
```

3. Enable the artifacts and chaos features of Halyard. Artifacts in Spinnaker merely reference any external resource, such as a remote file, a binary blob, and image, and so forth. The chaos feature allows us to utilize Chaos Monkey, the base form of which is built into Spinnaker by default.

```
hal config features edit --artifacts true
hal config features edit --chaos true
```

Add AWS Provider to Halyard

Now we need to also add AWS as another provider and account. Be sure to replace `<AWS_ACCOUNT_ID>` with the primary/managing account ID of your AWS account.

```
hal config provider aws account add aws-primary --account-id <AWS_ACCOUNT_ID> --assume-role role/spinnakerManaged

hal config provider aws enable
```

Add ECS Provider to Halyard

The last provider to enable is ECS. We'll add the `ecs-primary` account to Halyard and associate it with the `aws-primary` AWS account added above:

```
hal config provider ecs account add ecs-primary --aws-account aws-primary

hal config provider ecs enable
```

Use Distributed Deployment

We also need to ensure Halyard deploys Spinnaker in a distributed fashion among our Kubernetes cluster. Without this step, the default configuration is to deploy Spinnaker onto the local machine.

```
hal config deploy edit --type distributed --account-name kubernetes-master
```

Error: `kubectl` not installed, or can't be found by Halyard.

If you get such an error when issuing the distributed deployment command above, it likely means Halyard just needs to be restarted. Simply issue the `hal shutdown` command to stop the Halyard daemon, then retry the deployment edit command again, which will automatically restart Halyard before executing.

Use S3 for Persistent Storage

Let's tell Spinnaker to use AWS S3 for storing persistent data (in this case, creating a small S3 bucket). Issue the following command by replacing <AWS_ACCESS_KEY_ID> with any AWS access key that has full S3 service privileges.

```
hal config storage s3 edit --access-key-id <AWS_ACCESS_KEY_ID> --secret-access-key --region us-west-2

hal config storage edit --type s3
```

Create Kubernetes Worker Nodes

Now we'll launch some AWS EC2 instances which will be our worker nodes for our Kubernetes cluster to manage.

1. Download this `amazon-eks-nodegroup.yml` template file.

```
curl -O https://d3079gxvs8ayeg.cloudfront.net/templates/amazon-eks-nodegroup.yaml
```

2. Issue the following command to use the template and create your worker node collection.

```
aws cloudformation deploy --stack-name spinnaker-eks-nodes --template-file amazon-eks-nodegroup.yaml \
--parameter-overrides NodeInstanceProfile=$SPINNAKER_INSTANCE_PROFILE_ARN \
NodeInstanceType=t2.large ClusterName=$EKS_CLUSTER_NAME NodeGroupName=spinnaker-cluster-nodes ClusterControlPlaneSecurityGroup=$CONTROL_PLANE_SG \
Subnets=$SUBNETS VpcId=$VPC_ID --capabilities CAPABILITY_NAMED_IAM
```

3. To connect up our newly-launched worker instances with the Spinnaker cluster we previously deployed we need to create a new `~/.kube/aws-auth-cm.yaml` file. Paste the following text into `aws-auth-cm.yaml`, replacing `<AUTH_ARN>` with the `AUTH_ARN` variable created previously (Remember, you can use `echo $AUTH_ARN` to print to console).

Adjust Worker Nodes

The default template creates an auto-balancing collection of up to **3** worker nodes (instances). Additionally, the deployment command we'll be using below specifies `t2.large` instance types. As always, feel free to modify the `amazon-eks-nodegroup.yaml` or instance types to meet your needs.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: aws-auth
 namespace: kube-system
data:
 mapRoles: |
  - rolearn: <AUTH_ARN>
    username: system:node:{{EC2PrivateDNSName}}
    groups:
     - system:bootstrappers
     - system:nodes
```

4. Apply this newly created role mapping by issuing the following command.

```
kubectl apply -f ~/.kube/aws-auth-cm.yaml
```

5. Check the status of your Kubernetes nodes with `kubectl get nodes`. The `--watch` flag can be added to perform constant updates. Once all nodes have a `Ready` **STATUS** you're all set to deploy Spinnaker.

```
kubectl get nodes
NAME                                        STATUS   ROLES    AGE    VERSION
ip-10-100-10-178.us-west-2.compute.internal Ready    <none>   2m     v1.10.3
ip-10-100-10-210.us-west-2.compute.internal Ready    <none>   2m     v1.10.3
ip-10-100-11-239.us-west-2.compute.internal Ready    <none>   2m     v1.10.3
```

Deploy Spinnaker

1. Start by listing the current Spinnaker versions with `hal version list`:

```
hal version list
```

Handling `hal deploy apply` Errors

In some cases you may experience a deployment error, particularly when trying your first Spinnaker deployment. Often the console output is quite vague, so the best course of action is to check your Spinnaker/Halyard log files. Typically these are located in `/var/log/spinnaker` and `/var/log/spinnaker/halyard`. Since Halyard runs on Java, logs let you see the entire Java error stack trace, rather than the basic (and often useless) error name.

Profile-related `IndexOutOfBoundsException`

With recent Halyard/Spinnaker versions there's a known bug that you may experience in which an `IndexOutOfBoundsException` occurs during deployment when using the AWS provider. The cause usually seems to be that Halyard is assuming an explicit `region` value in the YAML configuration file for the AWS account being used. Even though the `aws` block in the config has a `defaultRegions` key, that seems to be ignored, which can cause this error. The current solution is to manually edit the primary AWS account and explicitly set the `region` value, which should solve the issue and allow you to run a Spinnaker deployment.

2. Start by listing the currea Specify the version you wish to install with the `--version` flag below. We'll be using the latest at the time of writing, `1.9.2`.

```
hal config version edit --version 1.9.2
```

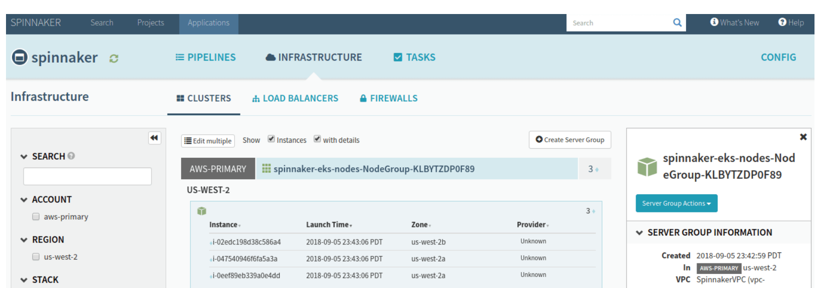
3. Now use `hal deploy apply` to deploy Spinnaker using all the configuration settings we've previously applied. This will go about distributing Spinnaker in your EKS/Kubernetes cluster.

```
hal deploy apply
```

```
hal config provider aws account edit aws-primary --add-region
us-west-2
```

That's it, you should now have a Spinnaker deployment up and running on a Kubernetes cluster, using EKS and EC2 worker node instances! Issue the `hal deploy connect` command to provide port forwarding on your local machine to the Kubernetes cluster running Spinnaker, then open http://localhost:9000 to make sure everything is up and running.

Select the `spinnaker` app and you should see your `aws-primary` account with a `spinnaker-eks-nodes-NodeGroup` containing your three EC2 worker node instances.



Next Steps

From here you can install and start using Chaos Monkey to run Chaos Experiments directly on your Kubernetes cluster worker nodes! Check out our How to Install Chaos Monkey tutorial to get started.

THE SIMIAN ARMY

Overview and Resources

The **Simian Army** is a suite of failure-inducing tools designed to add more capabilities beyond Chaos Monkey. While Chaos Monkey solely handles termination of random instances, Netflix engineers needed additional tools able to induce other types of failure. Some of the Simian Army tools have fallen out of favor in recent years and are deprecated, but each of the members serves a specific purpose aimed at bolstering a system's failure resilience.

In this chapter we'll jump into each member of the Simian Army and examine how these tools helped shape modern Chaos Engineering best practices. We'll also explore each of the Simian Chaos Strategies used to define which Chaos Experiments the system should undergo. Lastly, we'll plunge into a short tutorial walking through the basics of installing and using the Simian Army toolset.

Simian Army Members

Each Simian Army member was built to perform a small yet precise Chaos Experiment. Results from these tiny tests can be easily measured and acted upon, allowing you and your team to quickly adapt. By performing frequent, intentional failures within your own systems, you're able to create a more fault-tolerant application.

Active Simians

In addition to Chaos Monkey, the following simian trio are the only Army personnel to be publicly released, and which remain available for use today.

Janitor Monkey - Now Swabbie

Janitor Monkey also seeks out and disposes of unused resources within the cloud. It checks any given resource against a set of configurable rules to determine if its an eligible candidate for cleanup. Janitor Monkey features a number of configurable options, but the default behavior looks for resources like orphaned (non-auto-scaled) instances, volumes that are not attached to an instance, unused auto-scaling groups, and more.

Have a look at Using Simian Army Tools for a basic guide configuring and executing Janitor Monkey experiments.

A magnificent Janitor Monkey

Update: Swabbie is the Spinnaker service that replaces the functionality provided by Janitor Monkey. Find out more in the official documentation.

Conformity Monkey - Now Part of Spinnaker

The **Conformity Monkey** is similar to **Janitor Monkey** -- it seeks out instances that don't conform to predefined rule sets and shuts them down. Here are a few of the non-conformities that Conformity Monkey looks for.

> Auto-scaling groups and their associated elastic load balancers that have mismatched availability zones.
>
> Clustered instances that are not contained in required security groups.
>
> Instances that are older than a certain age threshold.

Conformity Monkey capabilities have also been rolled into Spinnaker. More info on using Conformity Monkey can be found under Using Simian Army Tools.

Security Monkey

Security Monkey was originally created as an extension to Conformity Monkey, and it locates potential security vulnerabilities and violations. It has since broken off into a self-contained, standalone, open-source project. The current 1.X version is capable of monitoring many common cloud provider accounts for policy changes and insecure configurations. It also ships with a single-page application web interface.

Inactive/Private Simians

This group of simians were either been deprecated or were never publicly released.

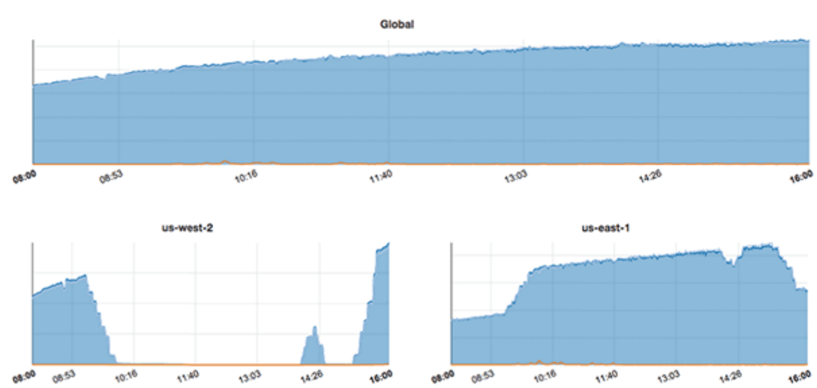
Chaos Gorilla

AWS Cloud resources are distributed around the world, with a current total of 18 geographic **Regions**. Each region consists of one or more **Availability Zones**. Each availability zone acts as a separated private network of redundancy, communicating with one another via fiber within their given region.

The **Chaos Gorilla** tool simulates the outage of entire AWS availability zone. It's been successfully used by Netflix to verify that their service load balancers functioned properly and kept services running, even in the event of an availability zone failure.

Chaos Kong

While rare, it is not unheard of for an AWS region to experience outages. Though Chaos Gorilla simulates availability zone outages, Netflix later created **Chaos Kong** to simulate region outages. As Netflix discusses in their blog, running frequent Chaos Kong experiments prior to any actual regional outages ensured that their systems were able to successfully evacuate traffic from the failing region into a nominal region, without suffering any severe degradation.



Netflix Chaos Kong Experiment Chart -- **Courtesy Netflix***

Netflix Chaos Kong Experiment - Courtesy of Netflix

Latency Monkey

Latency Monkey causes artificial delays in RESTful client-server communications and while it proved to be a useful tool. However, as Netflix later discovered, this particular Simian could be somewhat difficult to wrangle at times. By simulating network delays and failures, it allowed services can be tested to see how they react when their dependencies slow down or fail to respond, but these actions also occasionally caused unintended effects within other applications.

While Netflix never publicly released the Latency Monkey code, and it eventually evolved into their Failure Injection Testing (FIT) service, which we discuss in more detail over here.

Doctor Monkey

Doctor Monkey performs instance health checks and monitors vital metrics like CPU load, memory usage, and so forth. Any instance deemed unhealthy by Doctor Monkey is removed from service.

Doctor Monkey is not open-sourced, but most of its functionality is built into other tools like Spinnaker, which includes a load balancer health checker, so instances that fail certain criteria are terminated and immediately replaced by new ones. Check out the How to Deploy Spinnaker on Kubernetes tutorial to see this in action!

10-18 Monkey

The **10-18 Monkey** (aka `l10n-i18n`) detects run time issues and problematic configurations within instances that are accessible across multiple geographic regions, and which are serving unique localizations.

Simian Chaos Strategies

The original Chaos Monkey was built to inject failure by terminating EC2 instances. However, this provides a limited simulation scope, so Chaos **Strategies** were added to the Simian Army toolset. Most of these strategies are disabled by default, but they can be toggled in the `SimianArmy/src/main/resources/chaos.properties` configuration file.

Network Traffic Blocker (Simius Quies)

Blocks network traffic by applying restricted security access to the instance. This strategy only applies to VPC instances.

Configuration Key

`simianarmy.chaos.blockallnetworktraffic`

Instance Shutdown (Simius Mortus)

Shuts down an EC2 instance.

Configuration Key

`simianarmy.chaos.shutdowninstance`

Network Traffic Blocker (Simius Quies)

instance. This strategy only applies to VPC instances.

Configuration Key

`simianarmy.chaos.blockallnetworktraffic`

EBS Volume Detachment (Simius Amputa)

Detaches all EBS volumes from the instance to simulate I/O failure.

Configuration Key

`simianarmy.chaos.detachvolumes`

Burn-CPU (Simius Cogitarius)

Heavily utilizes the instance CPU.

Configuration Key

`simianarmy.chaos.burncpu`

Fill Disk (Simius Plenus)

Attempts to fill the instance disk.

Configuration Key

`simianarmy.chaos.shutdowninstance`

Kill Processes (Simius Delirius)

Kills all Python and Java processes once every second.

Configuration Key

`simianarmy.chaos.killprocesses`

Null-Route (Simius Desertus)

Severs all instance-to-instance network traffic by null-routing the 10.0.0.0/8 network.

Configuration Key

`simianarmy.chaos.nullroute`

Fail DNS (Simius Nonomenius)

Prevents all DNS requests by blocking TCP and UDP traffic to port 53.

Configuration Key

`simianarmy.chaos.faildns`

Fail EC2 API (Simius Noneccius)

Halts all EC2 API communication by adding invalid entries to /etc/hosts.

Configuration Key

`simianarmy.chaos.failec2`

Fail S3 API (Simius Amnesius)

Stops all S3 API traffic by placing invalid entries in `/etc/hosts`.

Configuration Key

`simianarmy.chaos.fails3`

Fail DynamoDB API (Simius Nodynamus)

Prevents all DynamoCB API communication by adding invalid entries to `/etc/hosts`.

Configuration Key

`simianarmy.chaos.faildynamodb`

Network Corruption (Simius Politicus)

Corrupts the majority of network packets using a traffic shaping API.

Configuration Key

`simianarmy.chaos.networkcorruption`

Network Latency (Simius Tardus)

Delays all network packets by 1 second, plus or minus half a second, using a traffic shaping API.

Configuration Key

`simianarmy.chaos.networklatency`

Network Loss (Simius Perditus)

Drops a fraction of all network packets by using a traffic shaping API.

Configuration Key

`simianarmy.chaos.networkloss`

Using Simian Army Tools

Prerequisites

Install and Setup AWS CLI

Install the Java JDK

1. Start by creating an AWS Auto Scaling launch configuration.

```
aws autoscaling create-launch-configuration --launch-
configuration-name simian-lc --instance-type t2.micro
--image-id ami-51537029
```

2. Now use the generated `simian-lc` configuration to create an Auto Scaling Group.

```
aws autoscaling create-auto-scaling-group --auto-
scaling-group-name monkey-target --launch-
configuration-name simian-lc --availability-zones
us-west-2a --min-size 1 --max-size 2
```

3. *(Optional)* Check that the scaling group was successfully added.

```
aws autoscaling describe-auto-scaling-groups --auto-
scaling-group-names monkey-target --output json
```

```
{
    “AutoScalingGroups”: [
        {
            “AutoScalingGroupARN”: “arn:aws:autoscaling:us-west-2:123456789012:autoScalingGroup:918
a23bc-ea5a-4def-bc68-5356becfd35d:autoScalingGroupName/monkey-target”,
            “ServiceLinkedRoleARN”: “arn:aws:iam::123456789012:role/aws-service-role/autoscaling.
amazonaws.com/AWSServiceRoleForAutoScaling”,
            “TargetGroupARNs”: [],
            “SuspendedProcesses”: [],
            “DesiredCapacity”: 1,
            “Tags”: [],
            “EnabledMetrics”: [],
            “LoadBalancerNames”: [],
            “AutoScalingGroupName”: “monkey-target”,
            “DefaultCooldown”: 300,
            “MinSize”: 1,
            “Instances”: [
                {
                    “ProtectedFromScaleIn”: false,
                    “AvailabilityZone”: “us-west-2a”,
                    “InstanceId”: “i-0e47c9f0df5150263”,
                    “HealthStatus”: “Healthy”,
                    “LifecycleState”: “Pending”,
                    “LaunchConfigurationName”: “simian-lc”
                }
            ],
            “MaxSize”: 2,
            “VPCZoneIdentifier”: “”,
            “HealthCheckGracePeriod”: 0,
            “TerminationPolicies”: [
                “Default”
            ],
            “LaunchConfigurationName”: “simian-lc”,
            “CreatedTime”: “2018-09-13T03:43:13.503Z”,
            “AvailabilityZones”: [
                “us-west-2a”
            ],
            “HealthCheckType”: “EC2”,
            “NewInstancesProtectedFromScaleIn”: false
        }
    ]
}
```

4. *(Optional)* Add any additional, manually-propagated EC2 instances you might need, using the same `ami-51537029` image used for the auto-scaling group.

```
aws ec2 run-instances --image-id ami-51537029 --count
1 --instance-type t2.micro --key-name id_rsa
```

```
# OUTPUT
123456789012 r-0ade24933c15617ba
INSTANCES 0    x86_64   False  xen     ami-51537029
i-062b161f4a1cddbb7    t2.micro       id_rsa
2018-09-13T03:50:07.000Z      ip-172-31-30-145.us-
west-2.compute.internal       172.31.30.145          /
dev/sda1       ebs     True           subnet-27c73d43
hvmvpc-0967976d
```

5. *(Optional)* Attach any manually-created EC2 instances to the `monkey-target` auto-scaling group.

```
aws autoscaling attach-instances --instance-ids
i-062b161f4a1cddbb7 --auto-scaling-group-name monkey-
target
```

Receiving Email Notifications

1. *(Optional)* If you want to receive email notifications you'll need to add an email address identity to AWS Simple Email Service (SES).

> ### us-east-1 Region only
>
> At present, SimianArmy only attempts to send email notifications through the AWS us-east-1 region, regardless of configuration settings. Thus, be sure the recipient address is in the us-east-1 AWS region.

```
aws ses verify-email-identity --email-address me@example.
com --region us-east-1
```

2. Open your email client and click the verification link.

3. Verify the address was successfully added to the proper SES region.

```
aws ses list-identities --region=us-east-1
```

```
# OUTPUT
IDENTITIES    me@example.com
```

Configuration

1. Clone the SimianArmy GitHub repository into the local directory of your choice.

```
git clone git://github.com/Netflix/SimianArmy.git ~/
SimianArmy
```

2. *(Optional)* Modify the `client.properties` configuration to change AWS connection settings.

```
nano ~/SimianArmy/src/main/resources/client.properties
```

3. *(Optional)* Modify the `simianarmy.properties` configuration to change general SimianArmy behavior.

```
nano ~/SimianArmy/src/main/resources/simianarmy.
properties
```

4. *(Optional)* Modify the `chaos.properties` configuration to change Chaos Monkey's behavior.

```
nano ~/SimianArmy/src/main/resources/chaos.properties
```

   - By default, Chaos Monkey won't target AWS Auto Scaling Groups unless you explicitly enable them. If desired, enable the recently added `monkey-target` ASG by adding the following setting.

```
simianarmy.chaos.ASG.monkey-target.enabled = true
```

5. *(Optional)* Modify the `janitor.properties` configuration to change Janitor Monkey's behavior.

```
nano ~/SimianArmy/src/main/resources/janitormonkey.
properties
```

6. *(Optional)* If you opted to receive SES notifications, specify the recipient email address within each appropriate configuration file. The following example modifies the `conformity.properties` file.

```
nano ~/SimianArmy/src/main/resources/conformity.
properties
```

```
# The property below needs to be a valid email address to receive the summary email of Conformity Monkey
# after each run
simianarmy.conformity.summaryEmail.to = foo@bar.com

# The property below needs to be a valid email address to send notifications for Conformity monkey
simianarmy.conformity.notification.defaultEmail = foo@bar.com

# The property below needs to be a valid email address to send notifications for Conformity Monkey
simianarmy.conformity.notification.sourceEmail = foo@bar.com
```

Executing Experiments

Run the included Gradle Jetty server to build and execute the Simian Army configuration.

```
./gradlew jettyRun
```

After the build completes you'll see log output from each enabled Simian Army members, including **Chaos Monkey 1.X**.

Using Chaos Monkey 1.X

```
2018-09-11 14:31:06.625 - INFO  BasicChaosMonkey -
[BasicChaosMonkey.java:276] Group monkey-target [type ASG]
enabled [prob 1.0]

2018-09-11 14:31:06.625 - INFO  BasicChaosInstanceSelector
- [BasicChaosInstanceSelector.java:89] Group monkey-
target [type ASG] got lucky: 0.9183174043024381 >
0.16666666666666666

2018-09-11 14:31:06.626 - INFO  Monkey - [Monkey.java:138]
Reporting what I did...
```

This older version of Chaos Monkey uses probability to pseudo-randomly determine when an instance should be terminated. The output above shows that 0.918… exceeds the required chance of 1/6, so nothing happened. However, running ./gradlew jettyRun a few times will eventually result in a success. If necessary, you can also modify the probability settings in the chaos.properties file.

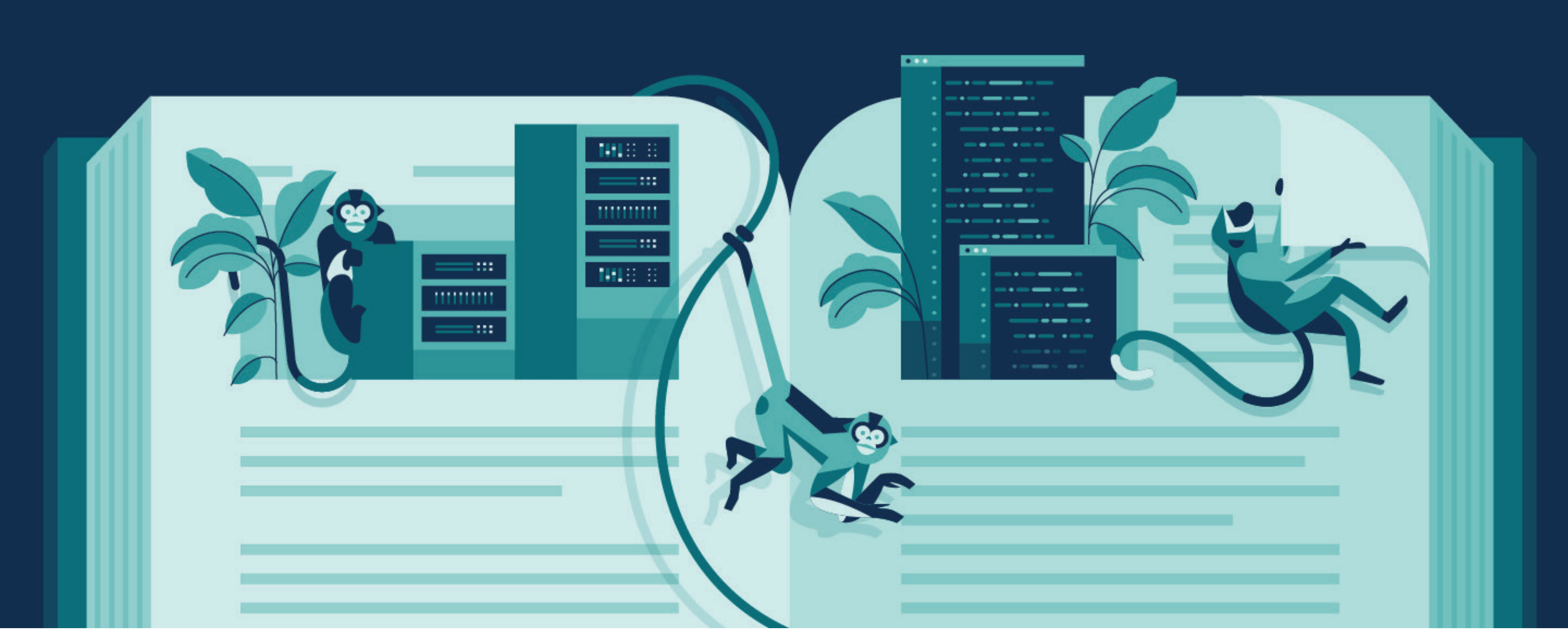
```
2018-09-11 14:33:06.625 - INFO  BasicChaosMonkey -
[BasicChaosMonkey.java:89] Group monkey-target [type ASG]
enabled [prob 1.0]

2018-09-11 14:33:06.625 - INFO  BasicChaosMonkey -
[BasicChaosMonkey.java:280] leashed ChaosMonkey prevented
from killing i-057701c3ab4f1e5a4 from group monkey-target
[ASG], set simianarmy.chaos.leashed=false
```

By default, the simianarmy.chaos.leashed = true property in chaos.properties prevents Chaos Monkey from terminating instances, as indicated in the above log output. However, changing this property to false allows Chaos Monkey to terminate the selected instance.

```
2018-09-11 14:33:56.225 - INFO  BasicChaosMonkey -
[BasicChaosMonkey.java:89] Group monkey-target [type ASG]
enabled [prob 1.0]

2018-09-11 14:33:56.225 - INFO  BasicChaosMonkey -
[BasicChaosMonkey.java:280] Terminated i-057701c3ab4f1e5a4
from group monkey-target [ASG]
```

Next Steps

Now that you've learned about the Simian Army, check out our Developer Tutorial to find out how to install and use the newer Chaos Monkey toolset. You can also learn about the many alternatives to Chaos Monkey, in which we shed light on tools and services designed to bring intelligent failure injection and powerful Chaos Engineering practices to your fingertips.



FOR ENGINEERS

Chaos Monkey Resources, Guides, and Downloads

We've collected and curated well **over 100** resources to help you with every aspect of your journey into Chaos Engineering. Learn about Chaos Engineering's origins and principles to shed light on what it's all about or dive right into one of the dozens of in-depth tutorials to get experimenting right away. You might also be interested in subscribing to some of the best Chaos Engineering blogs on the net or installing one of the many tools designed to inject failure into your applications, no matter the platform.

Chaos Engineering Best Practices & Principles

Without proper practices and principles Chaos Engineering becomes little more than unstructured Chaos. This section features a collection of the some of the most fundamental Chaos Engineering articles, practices, and principles ever devised.

- **Chaos Engineering: The History, Principles, and Practice**
- **FIT: Failure Injection Testing**
- **ChAP: Chaos Automation Platform**
- **Chaos Engineering - O'Reilly Media**
- **The Limitations of Chaos Engineering**
- **12 Factor Applications with Docker and Go**
- **Exploring Multi-level Weaknesses Using Automated Chaos Experiments**
- **Lineage Driven Failure Injection**
- **Failure Testing for Your Private Cloud**
- **Chaos Monkey for the Enterprise Cloud**
- **Chaos Engineering 101**
- **Chaos Monkey for Fun and Profit**
- **Chaos Monkey: Increasing SDN Reliability Through Systematic Network Destruction**
- **Agile DevOps: Unleash the Chaos Monkey**
- **Automating Failure Testing Research at Internet Scale**
- **The Netflix Simian Army**
- **Chaos Monkey Whitepaper**
- **Your First Chaos Experiment**

Chaos Engineering Blogs

One-off articles and tutorials have their place, but staying abreast of the latest Chaos Engineering news and technologies requires a constant drip of relevant information. The following blogs and community sites provide some of the most up-to-date SRE and Chaos Engineering content on the web.

- Gremlin Blog
- The Netflix Tech Blog
- Microsoft Azure Blog
- Spinnaker Blog
- AWS Open Source Blog
- SRE Weekly Newsletter
- LaunchDarkly Blog
- Coding Horror Blog
- Hut 8 Labs Blog

Chaos Engineering Community & Culture

A day doesn't go by without multiple people joining the Chaos Engineering Slack Channel! It's an exciting time to hop onto the Chaos Engineering train, but that journey wouldn't be nearly as interesting without the incredible culture and community that has built up around Chaos Engineering. This collection of resources contains just some of the many awesome aspects of the Chaos Engineering community.

- **The Chaos Engineering Slack Channel**
- **Chaos Engineering - Companies, People, Tools & Practices**
- **Chaos Conf - The Chaos Engineering Community Conference**
- **Gremlin Community**
- **Inside Azure Search: Chaos Engineering**
- **Breaking Things on Purpose**
- **Planning Your Own Chaos Day**
- **Business Continuity Plan & Disaster Recovery is Too Old**
- **Kafka in a Nutshell**
- **Can Spark Streaming survive Chaos Monkey?**
- **The Cloudcast #299 - The Discipline of Chaos Engineering**
- **Who is this Chaos Monkey and why did he crash my server?**
- **Netflix Chaos Monkey Upgraded**
- **Chaos Monkey and Resilience Testing - Insights From the Professionals**
- **Bees And Monkeys: 5 Cloud Lessons NAB Learned From AWS**
- **Working with the Chaos Monkey**
- **You've Heard of the Netflix Chaos Monkey? We Propose, for Cyber-Security, an "Infected Monkey"**
- **Building Your own Chaos Monkey**
- **Automated Failure Testing**
- **Active-Active for Multi-Regional Resiliency**
- **Post-Mortem of October 22, 2012 AWS Degradation**
- **Netflix to Open Source Army of Cloud Monkeys**
- **Chaos Engineering Upgraded**
- **When it Comes to Chaos, Gorillas Before Monkeys**
- **Continuous Chaos: Never Stop Iterating**
- **Oh Noes! The Sky is Falling!**

Chaos Engineering Talks

As more people take up the banner of Chaos Engineering we're treated to even more incredible presentations from some of the most brilliant minds in the field. We've gathered a handful of the most ground-breaking and influential of these talks below.

- **Intro to Chaos Engineering**
- **Testing In Production, The Netflix Way**
- **The Case for Chaos: Thinking About Failure Holistically**
- **1000 Actors, One Chaos Monkey and… Everything OK**
- **Orchestrating Mayhem Functional Chaos Engineering**
- **Using Chaos Engineering to Level Up Kafka Skills**
- **Chaos Engineering for vSphere**
- **Unbreakable: Learning to Bend But Not Break at Netflix**
- **Automating Chaos Experiments in Production**
- **Resiliency Through Failure - Netflix's Approach to Extreme Availability in the Cloud**

Chaos Engineering Tools

Proper tooling is the backbone of thoughtful and well-executed Chaos Engineering. As we showed in the Chaos Monkey Alternatives chapter, no matter what technology or platform you prefer, there are tools out there to begin injecting failure and to help you learn how to create more resilient systems.

- **Awesome Chaos Engineering: A curated list of Chaos Engineering resources**
- **Gremlin: Break things on purpose.**
- **Chaos Toolkit: Chaos Engineering Experiments, Automation, & Orchestration**
- **Marathon: A container orchestration platform for Mesos and DC/OS**
- **WazMonkey: A simple tool for testing resilience of Windows Azure cloud services**
- **Pumba: Chaos testing and network emulation tool for Docker**
- **Docker Simian Army: Docker image of Netflix's Simian Army**
- **Docker Chaos Monkey: A Chaos Monkey system for Docker Swarm**
- **Chaos Monkey - Elixir: Kill Elixir processes randomly**
- **Chaos Spawn: Chaotic spawning for elixir**
- **GoogleCloudChaosMonkey: Google Cloud Chaos Monkey tool**
- **Chaos Toolkit- Google Cloud: Chaos Extension for the Google Cloud Engine platform**
- **Kube Monkey: An implementation of Netflix's Chaos Monkey for Kubernetes clusters**
- **Pod Reaper: Rule based pod killing kubernetes controller**
- **Powerful Seal: A powerful testing tool for Kubernetes clusters.**
- **Monkey Ops: Chaos Monkey for OpenShift V3.X**
- **GomJabbar: Chaos Monkey for your private cloud**
- **Toxiproxy: A TCP proxy to simulate network and system conditions for chaos and resiliency testing**
- **Chaos Lemur: An adaptation of the Chaos Monkey concept to BOSH releases**
- **Chaos Monkey: A resiliency tool that helps applications tolerate random instance failures**
- **Vegeta: HTTP load testing tool and library.**
- **Simian Army: Tools for keeping your cloud operating in top form**
- **Security Monkey: Monitors AWS, GCP, OpenStack, and GitHub orgs for assets and their changes over time**
- **The Chaos Monkey Army**
- **Chaos Monkey Engine: A Chaos Engineering swiss army knife**
- **10 open-source Kubernetes tools for highly effective SRE and Ops Teams**
- **Chaos Lambda: Randomly terminate ASG instances during business hours**
- **Byte Monkey: Bytecode-level fault injection for the JVM**
- **Blockade: Docker-based utility for testing network failures and partitions in distributed applications**
- **Muxy: Chaos engineering tool for simulating real-world distributed system failures**
- **Chaos Hub: A Chaos Engineering Control Plane**

- **Chaos Toolkit Demos**
- **OS Faults: An OpenStack fault injection library**
- **Curated list of resources on testing distributed systems**
- **Anarchy Ape: Fault injection tool for Hadoop clusters**
- **Hadoop Killer: A process-based fault injector for Java**

Chaos Engineering Tutorials

Before you can swim in the deep end of Chaos Engineering you'll need to start by getting your feet wet. We've accumulated a number of tutorials covering just about every platform and technology you could be using, all of which provide a great jumping-off point to start executing Chaos Experiments in your own systems.

- **How to Install and Use Gremlin on Ubuntu 16.04**
- **How to Deploy - Chaos Monkey**
- **4 Chaos Experiments to Start With**
- **How to Setup and Use the Gremlin Datadog Integration**
- **Chaos Engineering and Mesos**
- **Create Chaos and Failover Tests for Azure Service Fabric**
- **Induce Chaos in Service Fabric Clusters**
- **How to Install and Use Gremlin with Kubernetes**
- **Chaos Testing for Docker Containers**
- **How to Install and Use Gremlin with Docker on Ubuntu 16.04**
- **Pumba - Chaos Testing for Docker**
- **Running Chaos Monkey on Spinnaker/Google Compute Engine**
- **Observing the Impact of Swapping Nodes in GKE with Chaos Engineering**
- **Chaos Monkey for Spring Boot**
- **How to Install and Use Gremlin on CentOS 7**
- **Improve Your Cloud Native DevOps Flow with Chaos Engineering**
- **Chaos Experiment: Split Braining Akka Clusters**
- **Kubernetes Chaos Monkey on IBM Cloud Private**
- **Introduction to Chaos Monkey**
- **Using Chaos Monkey Whenever You Feel Like It**
- **SimianArmy Wiki**
- **Continuous Delivery with Spinnaker**
- **Sailing with Spinnaker on AWS**
- **Chaos on OpenShift Clusters**
- **Automating a Chaos Engineering Environment on AWS with Terraform**
- **Gremlin Gameday: Breaking DynamoDB**

CHAOS MONKEY ALTERNATIVES

Tools for Creating Chaos Outside of AWS

Chaos Monkey serves a singular purpose -- to randomly terminate instances. As discussed in Chaos Monkey and Spinnaker and The Pros and Cons of Chaos Monkey, additional tools are required when using Chaos Monkey, in order to cover the broad spectrum of experimentation and failure injection required for proper Chaos Engineering.

In this chapter, we'll explore a wide range of tools and techniques -- regardless of the underlying technologies -- that you and your team can use to intelligently induce failures while confidently building toward a more resilient system.

Apache

Perhaps the most prominent fault-tolerant tool for Apache is Cassandra, the NoSQL, performant, highly-scalable data management solution. Check out this talk by Christos Kalantzis, Netflix's Director of Engineering for a great example of how Chaos Engineering can be applied within Cassandra.

Hadoop

Hadoop's unique Distributed File System (HDFS) requires that the FileSystem Java API and shell access allow applications and the operating system to read and interact with the HDFS. Therefore, most of the Chaos Engineering tools that run on underlying systems can also be used for injecting failure into Hadoop. Check out the Linux section in particular.

Anarchy Ape

Anarchy Ape is an open-source tool primarily coded in Java that injects faults into Hadoop cluster nodes. It is capable of corrupting specific files, corrupting random HDFS blocks, disrupting networks, killing DataNodes or NameNodes, dropping network packets, and much more.

Anarchy Ape is executed via the ape command line tool. Check out the official repository for more details on installing and using Anarchy Ape on your Hadoop clusters.

Hadoop Killer

Hadoop Killer is an open-source tool written in Ruby that kills random Java processes on a local system. It can be installed using RubyGems and is configured via a simple YAML syntax.

```
kill:
  target : “MyProcess”
  max: 3
  probability: 20
  interval: 1
```

> `target`: The name of the process to kill.
>
> `max`: Maximum number of times to kill the process.
>
> `probability`: Percentage probability to kill the process during each attempt.
>
> `interval`: Number of seconds between attempts.

Have a look at the GitHub repository for the basic info on using Hadoop Killer.

Kafka

The primary fault injection tool explicitly built for Kafka is its built-in Trogdor test framework. Trogdor executes fault injection through a single-coordinator multi-agent process. Trogdor has two built-in fault types.

> ProcessStopFault: Stops the specified process by sending a SIGSTOP signal.
>
> NetworkPartitionFault: Creates an artificial network partition between nodes using iptables.

Each user agent can be assigned to perform a **Task**, which is an action defined in JSON and includes the full Java class, along with startMs and durationMs, which indicate the milliseconds since the Unix epoch for when to start and how long to run the Task, respectively. All additional fields are customized and Task-specific.

Here's a simple Task to trigger a NetworkPartitionFault by creating a network partition between node1 and node2.

```
{
    "class": "org.apache.kafka.trogdor.fault.
NetworkPartitionFaultSpec",
    "startMs": 1000,
    "durationMs": 30000,
    "partitions": ["node1", "node2"]
}
```

Check out the wiki documentation for more details on using Trogdor to inject faults and perform tests in your Kafka system.

Spark

Chaos Engineering with Spark comes down to the underlying platforms on which your application resides. For example, if you're using the Tensorframes wrapper to integrate Spark with your TensorFlow application then Gremlin's Failure as a Service solution can help you inject failure and learn how to create a more resilient application.

Spark Streaming applications also include built-in fault-tolerance. Each Resilient Distributed Dataset (RDD) that Spark handles is subject to loss prevention policies defined in the Fault-Tolerance Semantics documentation.

Lastly, Spark's built-in integration tests include a handful of fault injections like the NetworkFaultInjection.

Containers

There are an abundance of Chaos Monkey alternatives for container-based applications. Browse through the Chaos Monkey Alternatives - Docker and Chaos Monkey Alternatives - OpenShift chapters for many great solutions.

Docker

Check out Chaos Monkey Alternatives - Docker for details on using Pumba, Gremlin, Docker Chaos Monkey, and Docker Simian Army to inject chaos into your Docker containers.

OpenShift

Head over to the Chaos Monkey Alternatives - OpenShift chapter for information on utilizing Monkey Ops, Gremlin, and Pumba to run Chaos Experiments in OpenShift distributions.

Erlang VM

In addition to the Elixir-specific Chaos Spawn tool, this presentation by Pavlo Baron shows a real-time demo of a Chaos Experiment that injects failure into 1,000 parallel actors within Erlang VM.

Elixir

Chaos Spawn is an open-source tool written in Elixir that periodically terminates low-level processes. Based on Chaos Monkey, Chaos Spawn has limited capabilities but it is also quite easy to install and configure.

1. To install Chaos Spawn just add `chaos_spawn` to your `mix.exs` dependencies.

```
# mix.exs
defp deps do
  [
    { :chaos_spawn, "~> 0.8.0" },
    # ...
  ]
end
```

2. Within `mix.exs` you'll also need to add `chaos_spawn` to `applications`.

```
def application do
  applications: [:chaos_spawn, :phoenix, :phoenix_html, :logger]]
end
```

3. Add `use ChaosSpawn.Chaotic.Spawn` to any module that should be eligible to create targetable processes.

```
defmodule ChaosSpawn.Example.Spawn do
  use ChaosSpawn.Chaotic.Spawn

  def test do
    spawn fn ->
      IO.puts "Message sent"
      receive do
        _ -> IO.puts "Message received"
      end
    end
  end
end
```

4. You can also add `:chaos_spawn` configuration keys to your `config/config.exs` file.

```
# config/config.exs
# Milliseconds between spawn checks.
config :chaos_spawn, :kill_tick, 5000
# Per-tick probability of killing a targeted process (0.0 - 1.0).
config :chaos_spawn, :kill_probability, 0.25
# Valid time period (UTC) in which to kill processes.
config :chaos_spawn, :only_kill_between, {% raw %}{{12, 00, 00}, {16, 00, 00}}{% endraw %}
```

Have a look at the official GitHub repository for more info on using Chaos Spawn to inject Chaos in your Elixir applications.

Infrastructure

There are dozens of alternative tools to Chaos Monkey available for the most popular infrastructure technologies and platforms on the market. Have a look through Chaos Monkey Alternatives - Azure, Chaos Monkey Alternatives - Google Cloud Platform, and Chaos Monkey Alternatives - Kubernetes for many great options.

Azure

Read through our Chaos Monkey Alternatives - Azure chapter for guidance on how the Azure Search team created their own Search Chaos Monkey, along with implementing your own Chaos Engineering practices in Azure with Gremlin, WazMonkey, and Azure's Fault Analysis Service.

Google Cloud Platform

Check out Chaos Monkey Alternatives - Google Cloud Platform for details on using the simple Google Cloud Chaos Monkey tool, Gremlin's Failure as a Service, and the open-source Chaos Toolkit for injecting failure into your own Google Cloud Platform systems.

Kubernetes

A quick read of our Chaos Monkey Alternatives - Kubernetes chapter will teach you all about the Kube Monkey, Kubernetes Pod Chaos Monkey, Chaos Toolkit, and Gremlin tools, which can be deployed on Kubernetes clusters to execute Chaos Experiments and create more resilient applications.

On-Premise

In addition to the many tools features in the Azure, Google Cloud Platform , Kubernetes, Private Cloud, and VMware sections we're looking at a few network manipulation tools for injecting failure in your on-premise architecture.

Blockade

Blockade is an open-source tool written in Python that creates various network failure scenarios within distributed applications. Blockade uses Docker containers to perform actions and manage the network from the host system.

1. To get started with Blockade you'll need a Docker container image for Blockade to use. We'll use `ubuntu:trusty` so make sure it's locally installed.

```
sudo docker pull ubuntu:trusty
```

2. Install Blockade via `pip`.

```
pip install blockade
```

3. Verify the installation with blockade -h.

```
blockade -h
```

4. Blockade is configured via the blockade.yml file, which defines the containers and the respective commands that will be executed by that container. These command values can be anything you'd like (such as running an app or service), but for this example, we're just performing a ping on port 4321 of the first container.

```
nano blockade.yml
```

```
containers:
  c1:
    image: ubuntu:trusty
    command: /bin/sleep 300000
    ports: [4321]

  c2:
    image: ubuntu:trusty
    command: sh -c "ping $C1_PORT_4321_TCP_ADDR"
    links: ["c1"]

  c3:
    image: ubuntu:trusty
    command: sh -c "ping $C1_PORT_4321_TCP_ADDR"
    links: ["c1"]
```

5. To run Blockade and have it create the specified containers use the blockade up command.

```
blockade up
# OUTPUT
NODE            CONTAINER ID    STATUS  IP
NETWORK     PARTITION
c1              dcb76a588453    UP      172.17.0.2
NORMAL
c2              e44421cae80f    UP      172.17.0.4
NORMAL
c3              de4510131684    UP      172.17.0.3
NORMAL
```

6. Blockade grabs the log output from each container, which can be viewed via `blockade logs <container>`. Here we're viewing the last few lines of the `c2` log output, which shows it is properly pinging port `4321` on container `c1`.

```
blockade logs c2 | tail
# OUTPUT
64 bytes from 172.17.0.2: icmp_seq=188 ttl=64
time=0.049 ms
64 bytes from 172.17.0.2: icmp_seq=189 ttl=64
time=0.100 ms
64 bytes from 172.17.0.2: icmp_seq=190 ttl=64
time=0.119 ms
64 bytes from 172.17.0.2: icmp_seq=191 ttl=64
time=0.034 ms
```

7. Blockade comes with a handful of network manipulation commands, each of which can be applied to one or more containers.

   - `blockade duplicate <container1>[, <containerN>]`: Randomly generates duplicate packets.
   - `blockade fast <container>[, <containerN>]`: Reverts any previous modifications.
   - `blockade flaky <container>[, <containerN>]`: Randomly drops packets.
   - `blockade slow <container>[, <containerN>]`: Slows the network.
   - `blockade partition <container>[, <containerN>]`: Creates a network partition.

8. Run a test on `c2` to see how it impacts traffic. Here we're slowing `c2`.

```
blockade slow c2
blockade logs c2 | tail
```

```
# OUTPUT
64 bytes from 172.17.0.2: icmp_seq=535 ttl=64
time=86.3 ms
64 bytes from 172.17.0.2: icmp_seq=536 ttl=64
time=0.120 ms
64 bytes from 172.17.0.2: icmp_seq=537 ttl=64 time=116
ms
64 bytes from 172.17.0.2: icmp_seq=538 ttl=64
time=85.1 ms
```

We can see that the network has been slowed significantly, causing (relatively) massive delays to the majority of our ping requests.

> ### Clean Up
> If you need to clean up any containers created by Blockade just run the `blockade destroy` command.
>
> ```
> blockade destroy
> ```

Check out the official documentation for more details on using Blockade.

Toxiproxy

Toxiproxy is an open-source framework written in Go for simulating network conditions within application code. It is primarily intended for testing, continuous integration, and development environments, but it can be customized to support randomized Chaos Experiments. Much of Toxiproxy's integration comes from open-source client APIs, which make it easy for Toxiproxy to integrate with a given application stack.

As an API within your application Toxiproxy can accomplish a lot of network simulations and manipulations, but this example shows how to use Toxiproxy to disconnect all Redis connections during a Rails application test.

1. (Optional) Create a Rails application.

```
rails new toxidemo && cd toxidemo
```

2. Add the `toxiproxy` and `redis` gems to the `Gemfile`.

```
# Gemfile
gem 'redis'
gem 'toxiproxy'
```

3. Install all gems.

```
bundle install
```

4. Scaffold a new `Post` model in Rails.

```
rails g scaffold Post tags:string
```

5. For this test, we need to map both listener and upstream Redis ports for Toxiproxy. Add this to `config/boot.rb` to ensure it executes before connections are established.

```
# config/boot.rb
require 'toxiproxy'

Toxiproxy.populate([
  {
    name: "toxiproxy_test_redis_tags",
    listen: "127.0.0.1:22222",
    upstream: "127.0.0.1:6379"
  }
])
```

6. To create the `TagRedis` proxied instance we need to add the following to `config/environment/test.rb` so it is only created during test executions.

```
# config/environment/test.rb
TagRedis = Redis.new(port: 22222)
```

7. Add the following methods to `app/models/post.rb`, which explicitly calls the proxied `TagRedis` instance that is created above.

```ruby
# app/models/post.rb
def tags
  TagRedis.smembers(tag_key)
def

def add_tag(tag)
  TagRedis.sadd(tag_key, tag)
def

def remove_tag(tag)
  TagRedis.srem(tag_key, tag)
def

def tag_key
  “post:tags:#{self.id}”
end
```

8. Add the following test to `test/models/post_test.rb`.

```ruby
setup do
  @post = posts(:one)
end

test “should return empty array when tag redis is down while listing tags” do
  @post.add_tag “gremlins”

  Toxiproxy[/redis/].down do
    assert_equal [], @post.tags
  end
end
```

9. Migrate the database for the first time if you haven’t done so.

```
rails db:migrate
```

10. Open a second terminal window and start the Toxiproxy server so it’ll listen for connections.

```
toxiproxy-server

# OUTPUT

INFO[0000] API HTTP server starting    host=localhost
port=8474 version=2.1.3
```

11 Now run our test with `rake test`. You should see the `Redis::CannotConnectError` reported from Rails.

```
# OUTPUT

Error:

PostTest#test_should_return_empty_array_when_tag_
redis_is_down_while_listing_tags:

Redis::CannotConnectError: Error connecting to Redis
on 127.0.0.1:22222 (Errno::ECONNREFUSED)
```

This is because Toxiproxy successfully closed the Redis connection during the test. Looking at the output from the `toxiproxy-server` window confirms this.

```
# OUTPUT

INFO[0108] Started proxy
name=toxiproxy_test_redis_tags proxy=127.0.0.1:22222
upstream=127.0.0.1:6379

INFO[0110] Accepted client
client=127.0.0.1:49958 name=toxiproxy_test_redis_tags
proxy=127.0.0.1:22222 upstream=127.0.0.1:6379

INFO[0110] Terminated proxy
name=toxiproxy_test_redis_tags proxy=127.0.0.1:22222
upstream=127.0.0.1:6379

WARN[0110] Source terminated
bytes=4 err=read tcp 127.0.0.1:51080->127.0.0.1:6379:
use of closed network connection name=toxiproxy_test_
redis_tags

WARN[0110] Source terminated
bytes=54 err=read tcp 127.0.0.1:22222-
>127.0.0.1:49958: use of closed network connection
name=toxiproxy_test_redis_tags

INFO[0110] Started proxy
name=toxiproxy_test_redis_tags proxy=127.0.0.1:22222
upstream=127.0.0.1:6379

INFO[0110] Accepted client
client=127.0.0.1:49970 name=toxiproxy_test_redis_tags
proxy=127.0.0.1:22222 upstream=127.0.0.1:6379

WARN[0110] Source terminated
```

12. In this simple example this test can pass by altering the app/models/post.rb#tags method to rescue the Redis::CannotConnectError.

```
# app/models/post.rb
def tags
  TagRedis.smembers(tag_key)
rescue Redis::CannotConnectError
  []
end
```

Check out this Shopify blog post and the official repository for more information on setting up and using Toxiproxy in your own architecture.

OpenStack

As software for data center management OpenStack can universally employ virtually any infrastructure-based Chaos tool we've already covered. Additionally, the OS-Faults tool is an open-source library developed by the OpenStack team and written in Python that is designed to perform destructive actions within an OpenStack cloud. Actions are defined using technology- and platform-specific drivers. There are currently drivers for nearly every aspect of OpenStack's architecture including drivers for the cloud, power management, node discovering, services, and containers.

While using OpenStack will depend heavily on your particular use case, below is one example showing the basics to get started.

1. Install OS-Faults via pip.

```
pip install os-faults
```

2. Configuration is read from a JSON or YAML file (os-faults.{json,yaml,yml}). Here we'll create the /etc/os-faults.yml file and paste the following inside.

```yaml
cloud_management:
  driver: devstack
  args:
    address: 192.168.1.10
    username: ubuntu
    password: password
    private_key_file: ~/.ssh/developer
    slaves:
    - 192.168.1.11
    iface: eth1

power_managements:
- driver: libvirt
  args:
    connection_uri: qemu+unix:///system
```

The `cloud_management` block contains just a single driver to be used, with relevant args such as the IP and credentials. The `power_managements` block can contain a list of drivers.

3. Now inject some failure by using the command line API. Here we're performing a simple restart of the `redis` service.

```
os-inject-fault restart redis service
```

> You may need to specify the location of your configuration file by adding the `-c <config-path>` flag to CLI commands.

That's all there is to it. OS-Faults is capable of many types of Chaos Experiments including disconnecting nodes, overloading IO and memory, restarting and killing services/nodes, and much more. The official documentation has more details.

Private Cloud

Take a look at the Chaos Monkey Alternatives - Private Cloud chapter to see how to begin Chaos Engineering within your own private cloud architecture using GomJabbar, Gremlin, and Muxy.

VMware

Check out the Chaos Monkey Alternatives - VMware chapter to learn about the Chaos Lemur tool and Gremlin's own Failure as a Service solution, both of will inject failure into your VMware and other BOSH-managed virtual machines with relative ease.

Java

The tools you choose for implementing Chaos Engineering in Java will primarily be based on the technologies of your system architecture. However, in addition to the Maven and Spring Boot tools discussed below, you may also consider Namazu, which is an open-source fuzzy scheduler for testing distributed system implementations. Via a pseudo-randomized schedule, Namazu can attack the filesystem and IO, network packets, and Java function calls.

Namazu can be installed locally or via a Docker container. Additionally, the Namazu Swarm plugin allows multiple jobs to be paralleled via Docker swarms or Kubernetes clusters.

Check out the official GitHub repository for more details on Chaos Engineering your Java applications with Nazamu.

Maven

Since Maven is a build automation tool for Java applications performing Chaos Experiments in Maven-based projects is as easy as utilizing one of the Java-related Chaos Engineering tools we've detailed elsewhere. Check out our guides for Chaos Monkey for Spring Boot, Gremlin, Fabric8, and Chaos Lemur for some ways to inject failure in Maven- and Java-based systems.

Spring Boot

Exploring our Chaos Monkey for Spring Boot chapter will teach you about how to use Chaos Monkey for Spring Boot, Gremlin, and Fabric8 to execute Chaos Experiments against your Spring Boot applications.

OS

Regardless of the operating system you're using there is a slew of Chaos Monkey alternative technologies. Check out the notable selections in the Linux and Windows sections below for some quick tools to get your Chaos Engineering kicked into high gear.

Linux

Performing Chaos Engineering on Linux is a match made in heaven -- virtually every tool we've explored here in the Alternatives chapter and listed in our Resources - Tools section was designed for if not built with Unix/Linux.

Some standout alternatives for Linux include The Chaos Toolkit on Kubernetes, Gremlin's Failure as a Service on nearly every platform, Pumba on Docker, and Chaos Lemur on BOSH-managed platforms.

Windows

Like Linux, Windows Chaos Engineering depends on the platforms and architecture your organization is using. Much of the software we've covered in Resources - Tools and this entire Alternatives chapter can be applied to Windows-based systems.

Some particularly useful Chaos Monkey alternatives for Windows are the Fault Analysis Service for Azure Service Fabric, Gremlin's Failure as a Service for Docker, and Muxy for private cloud infrastructures.