
Canonical Chaos Engineering

Canonical Group Ltd

Mar 25, 2026

CONTENTS

1	In this documentation	1
2	Project and community	3
2.1	Tutorial	3
2.2	How-to guides	13
2.3	Reference	19
2.4	Explanation	19

IN THIS DOCUMENTATION

Tutorial

Start here: a hands-on introduction to Canonical Chaos Engineering platform for new users

[tutorial/](#) How-to guides

Step-by-step guides covering key operations and common tasks

[how-to/](#) **Reference**

Technical information - specifications, APIs, architecture

[reference/](#) Explanation

Discussion and clarification of key topics

[explanation/](#)

PROJECT AND COMMUNITY

Canonical Chaos Engineering platform is a member of the Ubuntu family. It's an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

- [Ubuntu Code of conduct](#)
- [Open a bug](#)
- [Contribute](#)

2.1 Tutorial

In these tutorials, we will deploy and run the Canonical Chaos Engineering platform.

These tutorials will introduce you to key concepts, tools, processes and operations, starting from your first installation to a cloud deployment. Along the way, they will give examples of good practice and pointers to much more detailed information.

Completing the tutorial is a strongly-recommended investment of time if you're new to Chaos Engineering - it will save you many more hours later on. Follow the tutorials steps in sequence; they take you on a learning journey through the product.

The tutorials have been tested with a variety of users. We make every effort to keep them up-to-date and ensure that they are reliable, but if you encounter any problems, please get in touch, as we want to help you complete the tutorials and have a frictionless experience with the Canonical Chaos Engineering platform.

2.1.1 Table of content

Getting started

In this tutorial, we will get you started with using the Canonical Chaos Engineering platform. You will learn how to deploy the platform using Terraform and Juju, create a Chaos infrastructure and conduct a simple Chaos Experiment simulating POD deletion to check if the System Under Test can recover from such fault.

To complete this tutorial, you will need a machine which meets the following requirements:

- 8GB of RAM
- 50GB of free disk space
- An Ubuntu (or another operating system which supports [snapd](#)) environment to run the commands

Note

All the code blocks in this tutorial should be executed from a Linux shell.

1. Install Canonical K8s

Install Canonical K8s and bootstrap it:

```
sudo snap install k8s --classic --channel=1.33-classic/stable
cat << EOF | sudo k8s bootstrap --file -
containerd-base-dir: /opt/containerd
disable-rbac: true
cluster-config:
  network:
    enabled: true
  dns:
    enabled: true
  local-storage:
    enabled: true
EOF
```

2. Bootstrap a Juju controller

Install Juju:

```
sudo snap install juju --channel=3.6/stable
```

Save the K8s credentials, we will need them later to bootstrap the Juju controller.

```
mkdir -p ~/.kube
sudo k8s config > ~/.kube/config
mkdir -p ~/.local/share/juju/
sudo k8s config > ~/.local/share/juju/credentials.yaml
```

Bootstrap a Juju controller

```
juju bootstrap k8s
```

3. Install Terraform

Install Terraform:

```
sudo snap install terraform --classic
```

4. Deploy the Canonical Chaos Engineering platform

On the host machine create a new directory called terraform:

```
mkdir terraform
```

Inside the newly created terraform directory create a versions.tf file:

```
cd terraform
cat << EOF > versions.tf
terraform {
  required_providers {
    juju = {
```

(continues on next page)

(continued from previous page)

```

    source = "juju/juju"
    version = ">= 0.14.0"
  }
}
EOF

```

Create a new Terraform module for the Charmed Litmus Control Plane:

```

cat << EOF > control-plane.tf
resource "juju_model" "charmed-chaos" {
  name = "charmed-chaos"
}

module "charmed-litmus" {
  source = "git::https://github.com/canonical/litmus-operators//terraform"

  model_uuid = juju_model.charmed-chaos.uuid
  depends_on = [juju_model.charmed-chaos]

  litmus_channel = "dev/edge"
  admin_password = var.admin_password
  charm_password = var.charm_password
}
EOF

```

Create a `variables.tf` file:

```

cat << EOF > variables.tf
variable "admin_password" {
  description = "The password to set for the Litmus admin user."
  type        = string
  sensitive   = true
}

variable "charm_password" {
  description = "The password to set for the Litmus charm bot user."
  type        = string
  sensitive   = true
}
EOF

```

Note

You can get a ready example by cloning [this Git repository](#). All necessary files are in the `examples/terraform/getting_started` directory.

Initialize the Juju Terraform provider:

```
terraform init
```

Apply your Terraform configuration:

```
terraform apply -auto-approve
```

Note

Terraform will prompt you for two passwords - for the admin user and for the charm user. For this tutorial use Litmus123! and Charm123! respectively.

The deployment process should take approximately 5-10 minutes.

Monitor the status of the deployment:

```
juju switch charmed-chaos
juju status --relations --watch 1s
```

The deployment is ready when all the charms are in the active/idle state.

Example:

```
ubuntu@host:~/terraform $ juju status --relations
```

Model	Controller	Cloud/Region	Version	SLA
charmed-chaos	k8s	k8s	3.6.14	

```

↳ unsupported 12:25:15+02:00

App
↳ Address      Version  Status  Scale  Charm                      Channel  Rev
litmus-auth    3.26.0  active   1      litmus-auth-k8s           dev/edge 16  10.
↳ 152.183.56   no
litmus-backend 3.26.0  active   1      litmus-backend-k8s       dev/edge 17  10.
↳ 152.183.61   no
litmus-chaoscenter 3.26.0  active   1      litmus-chaoscenter-k8s   dev/edge 32  10.
↳ 152.183.151 no    Ready at http://litmus-chaoscenter.charmed-chaos.svc.cluster.
↳ local:8185.
mongodb-k8s    6.0.24  active   3      mongodb-k8s              6/stable 117 10.
↳ 152.183.50   no

Unit          Workload  Agent  Address      Ports  Message
litmus-auth/0*  active   idle   10.1.0.59
litmus-backend/0*  active   idle   10.1.0.100
litmus-chaoscenter/0*  active   idle   10.1.0.114    Ready at http://litmus-
↳ chaoscenter.charmed-chaos.svc.cluster.local:8185.
mongodb-k8s/0    active   idle   10.1.0.133
mongodb-k8s/1    active   idle   10.1.0.237
mongodb-k8s/2*   active   idle   10.1.0.35    Primary.

Integration provider  Requirer  Interface
↳ Type  Message
litmus-auth:http-api  litmus-chaoscenter:auth-http-api  litmus_auth_http_api
↳ regular
litmus-auth:litmus-auth  litmus-backend:litmus-auth  litmus_auth
↳ regular
litmus-backend:http-api  litmus-chaoscenter:backend-http-api  litmus_backend_http_
↳ api regular

```

(continues on next page)

(continued from previous page)

mongodb-k8s:database ↪ regular	litmus-auth:database	mongodb_client	↪
mongodb-k8s:database ↪ regular	litmus-backend:database	mongodb_client	↪
mongodb-k8s:database-peers ↪ peer	mongodb-k8s:database-peers	mongodb-peers	↪
mongodb-k8s:ldap-peers ↪ peer	mongodb-k8s:ldap-peers	ldap-peers	↪
mongodb-k8s:status-peers ↪ peer	mongodb-k8s:status-peers	status-peers	↪
mongodb-k8s:upgrade-version-a ↪ peer	mongodb-k8s:upgrade-version-a	upgrade	↪

5. Deploy System Under Test (SUT)

In this tutorial we will use the [self-signed-certificates] charm as a SUT (System-Under-Test).

Create a new Terraform module for the SUT:

```
cat << EOF > sut.tf
resource "juju_model" "self-signed-certs-test" {
  name = "self-signed-certs-test"
}

module "self-signed-certificates" {
  source = "git::https://github.com/canonical/self-signed-certificates-operator//
↪terraform"

  model_uuid = juju_model.self-signed-certs-test.uuid
  depends_on = [juju_model.self-signed-certs-test]
}

module "litmus-infrastructure" {
  source = "git::https://github.com/canonical/litmus-operators//infrastructure/terraform"

  channel = "dev/edge"

  model_uuid = juju_model.self-signed-certs-test.uuid
  depends_on = [juju_model.self-signed-certs-test]
}

resource "juju_offer" "litmus-infrastructure-offer" {
  model_uuid      = juju_model.self-signed-certs-test.uuid
  application_name = module.litmus-infrastructure.app_name
  endpoints       = [module.litmus-infrastructure.endpoints.litmus-infrastructure]
}
EOF
```

Connect Litmus infrastructure to the Litmus Control Plane:

```
cat << EOF >> control_plane.tf
resource "juju_integration" "litmus-infrastructure" {
```

(continues on next page)

(continued from previous page)

```

model_uuid = juju_model.charmed-chaos.uuid

application {
    name      = module.charmed-litmus.chaoscenter_app_name
    endpoint  = module.litmus-infrastructure.endpoints.litmus-infrastructure
}

application {
    offer_url = juju_offer.litmus-infrastructure-offer.url
}
}

EOF

```

Fetch new modules:

```
terraform init
```

Apply new configuration:

```
terraform apply -auto-approve
```

You can monitor the status of the deployment by running:

```
juju switch self-signed-certs-test
juju status --relations --watch 1s
```

Successful deployment of the SUT should look similar to below:

```

ubuntu@host:~/terraform $ juju status
Model                Controller  Cloud/Region  Version  SLA          Timestamp
self-signed-certs-test k8s        k8s           3.6.14   unsupported   15:07:10+01:00

App                Version  Status  Scale  Charm          Channel
↪Rev  Address      Exposed  Message
litmus-infrastructure 7  10.152.183.49  no      active      1  litmus-infrastructure-k8s  dev/edge
↪586  10.152.183.244 no      active      1  self-signed-certificates  1/stable

Unit                Workload  Agent  Address  Ports  Message
litmus-infrastructure/0* active    idle    10.1.0.30  Ports  Message
self-signed-certificates/0* active    idle    10.1.0.189

Offer                Application  Charm          Rev
↪Connected Endpoint  Interface  Role
certificates         self-signed-certificates  self-signed-certificates  586  0/0
↪ certificates       tls-certificates  provider
litmus-infrastructure litmus-infrastructure  litmus-infrastructure-k8s  7    1/1
↪ litmus-infrastructure litmus-infrastructure  provider
send-ca-cert         self-signed-certificates  self-signed-certificates  586  0/0
↪ send-ca-cert       certificate_transfer  provider

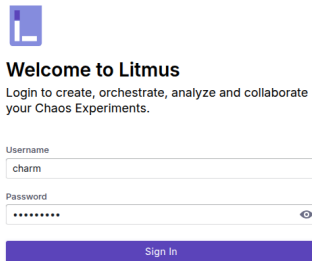
```

6. Run a Chaos experiment

Log in to the Litmus ChaosCenter

Retrieve the `litmus-chaoscenter/0` unit IP address from the `juju status` output. In the example above, this is `10.1.0.114`; yours will likely differ.

In your browser navigate to `http://{CHAOSCENTER_IP}:8185`. You should see the Litmus login page:

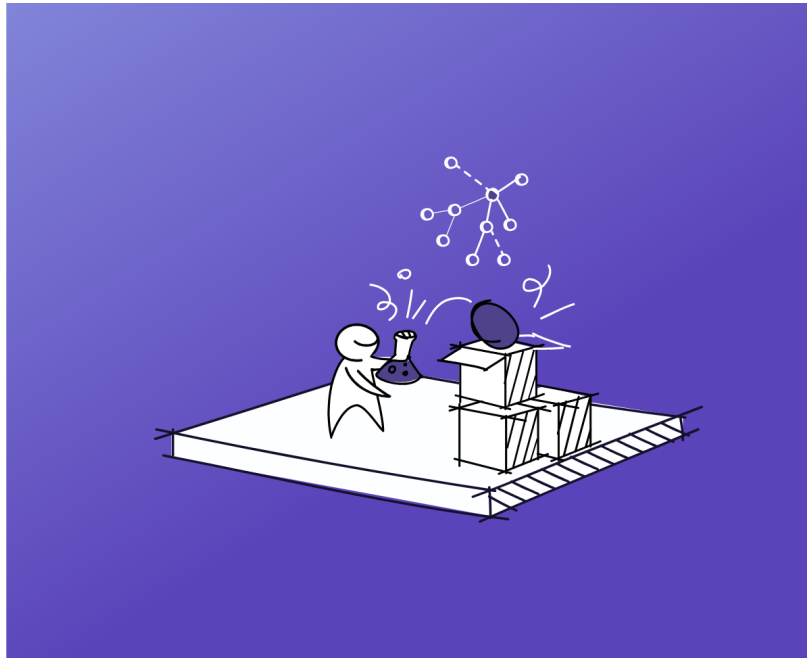


Welcome to Litmus
Login to create, orchestrate, analyze and collaborate your Chaos Experiments.

Username
charm

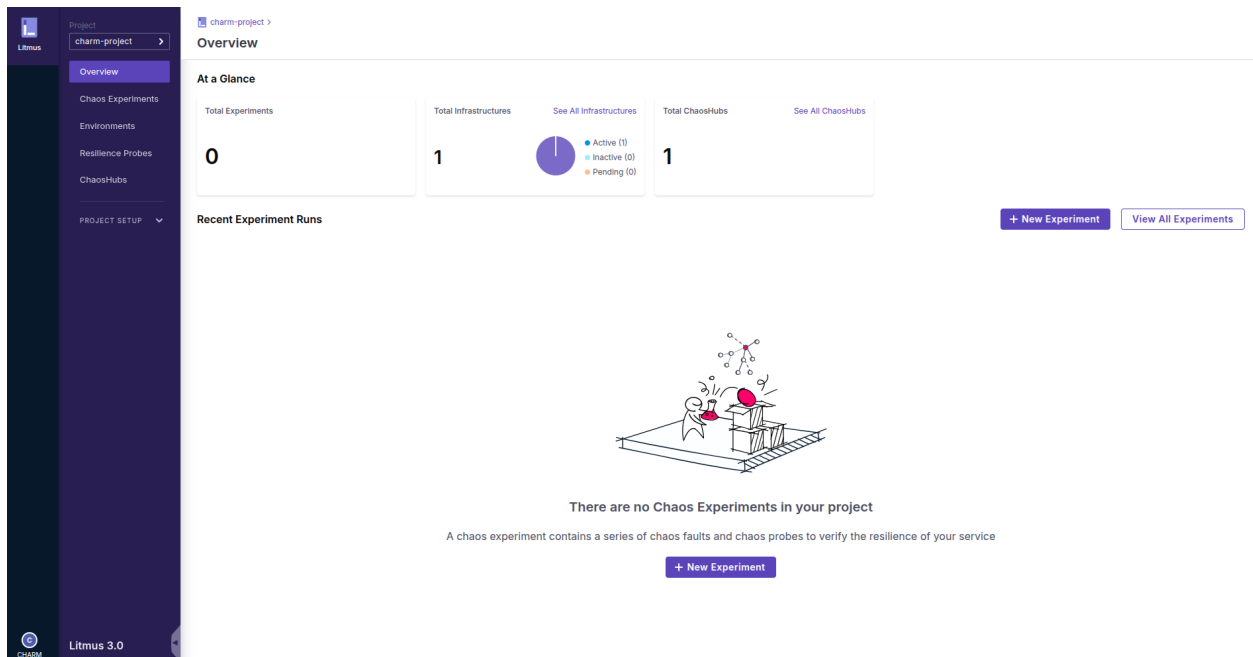
Password

Sign In



Log in to the Litmus ChaosCenter as user `charm` using the password set during the deployment (in this tutorial it is `Charm123!`).

After you successfully logged in, you should see the Litmus admin panel:



Overview

At a Glance

Total Experiments	Total Infrastructures	Total ChaosHubs
0	1	1

Recent Experiment Runs

There are no Chaos Experiments in your project

A chaos experiment contains a series of chaos faults and chaos probes to verify the resilience of your service

[+ New Experiment](#)

Note

Due to a bug in Litmus (<https://github.com/litmuschaos/litmus/issues/5224>) it is strongly recommended to log out immediately after the first login and log in again.

Define a Resilience Probe

In the Litmus ChaosCenter navigate to the **Resilience Probes** menu and click the **+ New Probe** button. Select the probe of type **Command** and configure it using the values below:

- Name: pod-up-probe
- Timeout: 10s
- Interval: 1s
- Attempt: 1
- Command: `kubectl -n self-signed-certs-test get pods | grep self-signed-certificates | grep Running | wc -l`
- Type: int
- Comparison Criteria: >
- Value: 0

A correctly configured probe should look like so:

The screenshot shows the Litmus ChaosCenter interface for configuring a probe. The breadcrumb is 'charm-project > Resilience Probes > pod-up-probe'. The probe name is 'pod-up-probe' with ID 'pod-up-probe'. It was created on 23 Mar 2026, 16:12 and last modified on 23 Mar 2026, 16:35. There is an 'Edit Probe' button. The 'Probe Configuration' tab is active, showing 'Probe Overview', 'Probe Properties', and 'Probe Details'.

Probe Overview

Probe Type

☒ Command Probe

The command probe allows developers to run shell commands and match the resulting output as part of the entry/exit criteria. The intent behind this probe was to allow users to implement a non-standard & imperative way of expressing their hypothesis. For example, the cmdProbe enables you to check for specific data within a database, parse the value out of a JSON blob being dumped into a certain path, or check for the existence of a particular string in the service logs.

Probe Properties

TIMEOUT	ATTEMPT	INTERVAL	POLLING INTERVAL	INITIAL DELAY	EVALUATION TIMEOUT	STOP ON FAILURE
10s	1	1s	N/A	N/A	N/A	N/A

Probe Details

COMMAND	SOURCE	COMPARATOR
kubectl -n self-signed-certs-test get pods grep self-signed-certificates grep Running wc -l	N/A	[object Object]

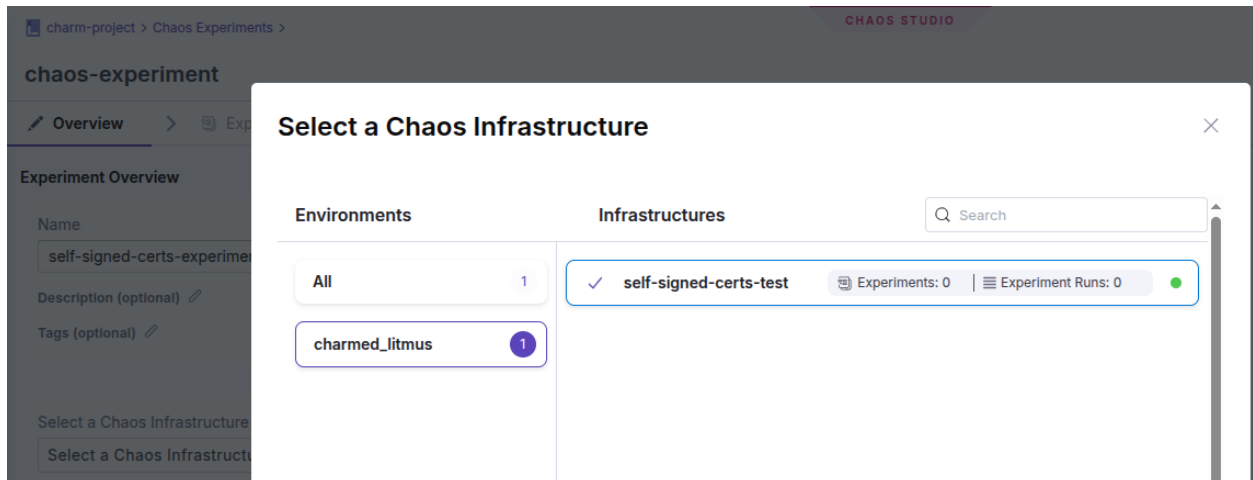
ON THIS PAGE

- Probe Overview
- Probe Properties
- Probe Details

Create a Chaos Experiment

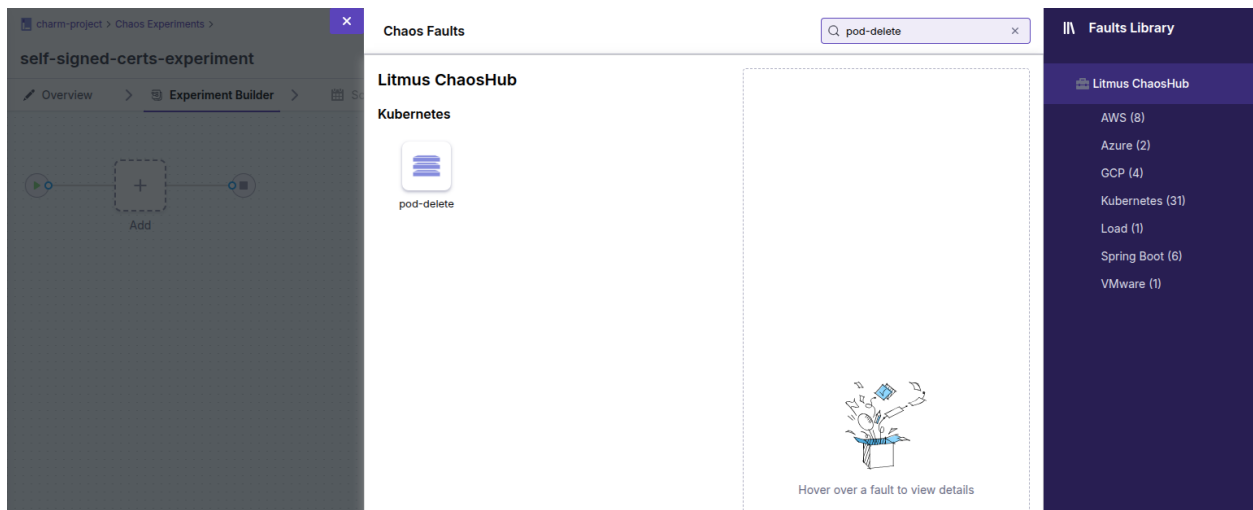
In the Litmus ChaosCenter navigate to the **Chaos Experiments** menu and click the **+ New Experiment** button.

Name the test and select a Chaos Infrastructure to use:



Start off building an experiment using Blank Canvas.

In the Experiment Builder click the Add button and add the pod-delete fault:



Configure the fault with the following values:

- App Kind: `statefulset`
- App Namespace: `self-signed-certs-test`
- App Label: `app.kubernetes.io/name=self-signed-certificates`


pod-delete-16o


[Apply Changes](#)
[Discard](#)

Target Application
Tune Fault
Probes

Provide target application details where you want to induce the chaos

App Kind

statefulset

App Namespace

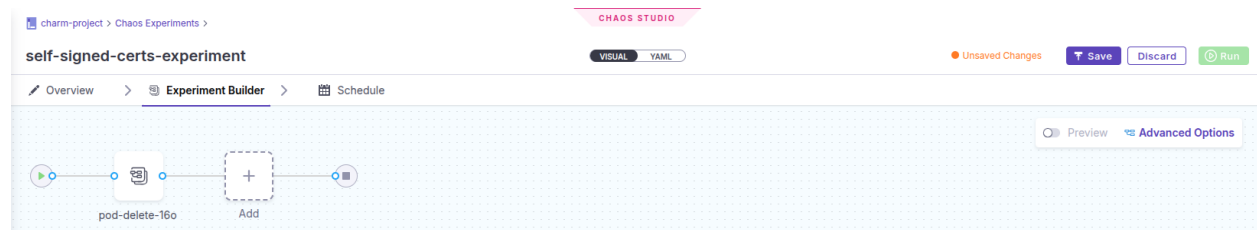
self-signed-certs-test

App Label

app.kubernetes.io/name=self-signed-certificates

In the Probes tab select the previously created pod-up-probe and confirm your choice by clicking the Add to Fault button. Select the End of Test (EOT) probe execution mode and apply changes.

At this point your experiment should look like this:



Save your changes by clicking the Save button in the top-right corner of the screen.

Run a Chaos Experiment

Click the Run button in the top-right corner of the screen to run the Chaos Experiment:



Running the experiment should take approximately 3 minutes.

When the experiment state changes from RUNNING to COMPLETED the run is done and the result is presented:

The screenshot displays the Canonical Chaos Engineering platform interface. At the top, the breadcrumb path is 'charm-project > Chaos Experiments > self-signed-certs-experiment'. The experiment title 'self-signed-certs-experiment' is followed by '(Execution ID: 3)' and a green 'COMPLETED' status. Below this, it states 'Hosted on Infrastructure: self-signed-certs-test' and 'Duration 23 Mar 2026, 16:37 - 16:40 @ 2m 5s'. A 'Resilience Score' of 100% is shown in a green box. A workflow diagram shows three steps: 'install-chaos-faults', 'pod-delete-16o', and 'cleanup-chaos-resources', all marked with green checkmarks. On the right, a panel for the 'Fault: pod-delete-16o' shows 'COMPLETED' status. It includes a 'Probe Result (Total 1)' with '1 Passed' and '0 Failed'. The 'Duration' is '23 Mar 2026, 16:38 - 16:39 @ 1m 2s' and the 'Namespace' is 'self-signed-certs-test'. Below this, there are tabs for 'Probes', 'Logs', and 'Fault Configuration'. The 'Logs' tab is active, showing a 'Console Logs' section with a list of log entries. The interface also has a sidebar with 'Run', 'Edit', 'Clone', and 'Download Manifest' buttons.

Congratulations! You have reached the end of this tutorial. For more information about the Canonical Chaos Engineering platform please see our other documents.

7. Destroy the environment

Destroy the Terraform deployment:

```
terraform destroy -auto-approve
```

Note

Terraform does not remove anything from the working directory. If needed, please clean up the terraform directory manually.

Destroy the Juju controller and all its models:

```
juju kill-controller k8s --no-prompt
```

Further reading

- [Litmus experiments](#)
- [Construct Chaos experiment YAML without ChaosCenter](#)
- [Schedule a Chaos experiment](#)
- [Save Chaos Scenarios as a Template](#)
- [Litmus ChaosHub](#)

2.2 How-to guides

These how-to guides cover key operations and processes in Canonical Chaos Engineering platform.

2.2.1 Day-1 operations

Day-1 operations cover the deployment and integration of Canonical Chaos Engineering platform.

Integrate Charmed Litmus with TLS

This how-to guide outlines the process of integrating Charmed Litmus with TLS in order to ensure encrypted communication across both internal service interactions and external client connections. TLS integration can be done as either Day 1 or Day 2 operation.

In this how-to we will use the [self-signed-certificates](#) charm to provide the necessary TLS certificates.

Note

For production deployments, we strongly discourage using self-signed TLS certificates. Instead, we recommend to use certificates signed by a trusted CA and safely stored, for example using a [Vault](#) charm.

1. Add self-signed-certificates to your Canonical Chaos Engineering platform Terraform module

Note

In this guide it is assumed that the Terraform module responsible for deploying the Charmed Litmus is named `charmed-litmus`. If you use a different name, make sure to update the code below.

Update your solution Terraform module (in this example named `main.tf`):

```
cat << EOF >> main.tf
module "self-signed-certificates" {
  source = "git::https://github.com/canonical/self-signed-certificates-operator//
  terraform"
  model = juju_model.charmed-chaos.name
}

resource "juju_integration" "litmus-auth-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name = module.charmed-litmus.auth_app_name
    endpoint = module.charmed-litmus.auth_tls_certificates_endpoint
  }

  application {
    name = module.self-signed-certificates.app_name
    endpoint = module.self-signed-certificates.provides.certificates
  }
}

resource "juju_integration" "litmus-backend-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name = module.charmed-litmus.backend_app_name
```

(continues on next page)

(continued from previous page)

```

    endpoint = module.charmed-litmus.backend_tls_certificates_endpoint
  }

  application {
    name      = module.self-signed-certificates.app_name
    endpoint  = module.self-signed-certificates.provides.certificates
  }
}

resource "juju_integration" "litmus-chaoscenter-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.chaoscenter_app_name
    endpoint  = module.charmed-litmus.chaoscenter_tls_certificates_endpoint
  }

  application {
    name      = module.self-signed-certificates.app_name
    endpoint  = module.self-signed-certificates.provides.certificates
  }
}

EOF

```

2. Apply the changes

Fetch the self-signed-certificates module:

```
terraform init
```

Apply the new configuration:

```
terraform apply -auto-approve
```

When the TLS integration is successfully completed, you will notice a change in the ChaosCenter URL printed in the `juju status` output. Namely, it will now indicate an `https` scheme. For example:

```

Unit                Workload Agent Address      Ports Message
(...)
litmus-chaoscenter/0* active   idle   10.1.194.255      Ready at https://litmus-
↳chaoscenter.charmed-chaos.svc.cluster.local:8185.
(...)

```

3. Example of a complete Terraform module including Charmed Litmus plus TLS integration

```

resource "juju_model" "charmed-chaos" {
  name = "charmed-chaos"
}

module "charmed-litmus" {

```

(continues on next page)

(continued from previous page)

```
source = "git::https://github.com/canonical/litmus-operators//terraform"

model      = juju_model.charmed-chaos.name
depends_on = [juju_model.charmed-chaos]
}

module "self-signed-certificates" {
  source = "git::https://github.com/canonical/self-signed-certificates-operator//
↪terraform"
  model  = juju_model.charmed-chaos.name
}

resource "juju_integration" "litmus-auth-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.auth_app_name
    endpoint  = module.charmed-litmus.auth_tls_certificates_endpoint
  }

  application {
    name      = module.self-signed-certificates.app_name
    endpoint  = module.self-signed-certificates.provides.certificates
  }
}

resource "juju_integration" "litmus-backend-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.backend_app_name
    endpoint  = module.charmed-litmus.backend_tls_certificates_endpoint
  }

  application {
    name      = module.self-signed-certificates.app_name
    endpoint  = module.self-signed-certificates.provides.certificates
  }
}

resource "juju_integration" "litmus-chaoscenter-tls" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.chaoscenter_app_name
    endpoint  = module.charmed-litmus.chaoscenter_tls_certificates_endpoint
  }

  application {
    name      = module.self-signed-certificates.app_name
    endpoint  = module.self-signed-certificates.provides.certificates
  }
}
```

(continues on next page)

(continued from previous page)

```
}
```

Enable ingress for Charmed Litmus

This how-to guide outlines the process of enabling ingress for Charmed Litmus. Enabling ingress can be done as both Day 1 and Day 2 operation.

In this how-to we will use the [Traefik Kubernetes Charmed Operator](#) as an ingress controller.

To complete this guide, your Kubernetes cluster will need a LoadBalancer with at least 1 available IP address.

1. Add Traefik to your Canonical Chaos Engineering platform Terraform module

Note

In this guide it is assumed that the Terraform module responsible for deploying the Charmed Litmus is named `charmed-litmus`. If you use different name, please make sure to update the code below.

Update your solution Terraform module (here it's named `main.tf`):

```
cat << EOF >> main.tf
module "traefik" {
  source = "git::https://github.com/canonical/traefik-k8s-operator//terraform"
  model  = juju_model.charmed-chaos.name
  channel = "latest/stable"
}

resource "juju_integration" "litmus-chaoscenter-traefik" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.chaoscenter_app_name
    endpoint  = module.charmed-litmus.chaoscenter_ingress_endpoint
  }

  application {
    name      = module.traefik.app_name
    endpoint  = module.traefik.endpoints.traefik_route
  }
}

EOF
```

2. Apply the changes

Fetch the traefik module:

```
terraform init
```

Apply new configuration:

```
terraform apply -auto-approve
```

Successful integration is indicated by the change of the ChaosCenter URL printed in `juju status` output. New address of the ChaosCenter should point to the IP address at which Traefik is served. Example:

Unit	Workload	Agent	Address	Ports	Message
(...)					
litmus-chaoscenter/0*	active	idle	10.1.194.214		Ready at http://10.0.0.3:8185.
(...)					
traefik/0*	active	idle	10.1.194.236		Serving at 10.0.0.3

3. Example of a complete solution Terraform module including Charmed Litmus integrated with Traefik

```
resource "juju_model" "charmed-chaos" {
  name = "charmed-chaos"
}

module "charmed-litmus" {
  source = "git::https://github.com/canonical/litmus-operators//terraform"

  model      = juju_model.charmed-chaos.name
  depends_on = [juju_model.charmed-chaos]
}

module "traefik" {
  source = "git::https://github.com/canonical/traefik-k8s-operator//terraform"
  model  = juju_model.charmed-chaos.name
  channel = "latest/stable"
}

resource "juju_integration" "litmus-chaoscenter-traefik" {
  model = juju_model.charmed-chaos.name

  application {
    name      = module.charmed-litmus.chaoscenter_app_name
    endpoint  = module.charmed-litmus.chaoscenter_ingress_endpoint
  }

  application {
    name      = module.traefik.app_name
    endpoint  = module.traefik.endpoints.traefik_route
  }
}
```

2.2.2 Day-2 operations

Day-2 operations cover the management and maintenance of Canonical Chaos Engineering platform over its lifecycle.

2.3 Reference

This section provides detailed reference material for Canonical Chaos Engineering platform.

2.3.1 Table of content

Known limitations

Here we list the known limitations of the Canonical Chaos Engineering platform in its current state.

These are wide-scoped missing features (or products) that we would like to work on eventually.

For a list of known issues, bugs and feature requests that are currently being worked on or being tracked, refer to these issue trackers:

- <https://github.com/canonical/chaos-engineering/issues>
- <https://github.com/canonical/litmus-operators/issues>
- <https://github.com/litmuschaos/litmus/issues>

Chaos testing for machine clouds

At the moment, the Canonical Chaos Engineering platform focuses on chaos-testing for k8s environments. We are currently not looking into charmed VM cloud chaos testing; but that is definitely an interesting avenue for future development. Thoughts on the matter? Get in touch!

Charmed Litmus execution plane

At the moment, the Charmed Litmus project only wraps the Control plane. Charming the Execution plane is something we'll be looking into in the near future, so stay tuned!

TLS integration for the Litmus Control plane

Due to an [upstream bug](#), at the moment the Litmus Control plane cannot talk over TLS to mongodb. The `mongodb-k8s` charm's quite tolerant TLS integration approach means that even if MongoDB has a `tls-certificates` relation, it will not reject the Litmus components' traffic, but the traffic will nonetheless be unencrypted. So beware of this issue, and keep an eye on the upstream issue for when a fix will eventually be shipped.

Admin password syncing for the Litmus Control plane

At the moment, Litmus doesn't wipe the MongoDB tables upon teardown, which means that if you change the admin password and forget it, the only way to 'reset' it is manually deleting the tables in MongoDB or redeploying the `mongodb-k8s` charm altogether. So beware that even if you redeploy the Control plane, the user settings will be persisted in the database. We're likely going to use juju-backed secret storage to manage the user credentials in an upcoming release, as admin credential management is a key aspect of implementing the Litmus execution plane, which we're going to work on soon (hopefully, delivered in 2026 Q2).

2.4 Explanation

The following explanations provide context and clarification on key-topics related to the use and configuration of Canonical Chaos Engineering platform.