CERN openlab

Improving Kubernetes Service Availability Through Chaos

Nivedita Prasad

Supervisors:

Ricardo Rocha, Spyridon Trigazis

15/09/2022

Let's begin!!!

- During the development process, different challenges and difficulties may arise. And handling those challenges during the development is far better than after the product is worldwide and is ready to serve. If any failure arises at that time, then it is not only a hectic process but also a costly one.

- For example, Facebook went down for 6 hours on Oct 5th, 2021, not only Facebook but WhatsApp, Instagram, and Messenger. That outage cost $160 million to the global economy (source). The question is, what could be done to fix it before this kind of situation arises? Answer - Chaos Engineering

What is Chaos Engineering?

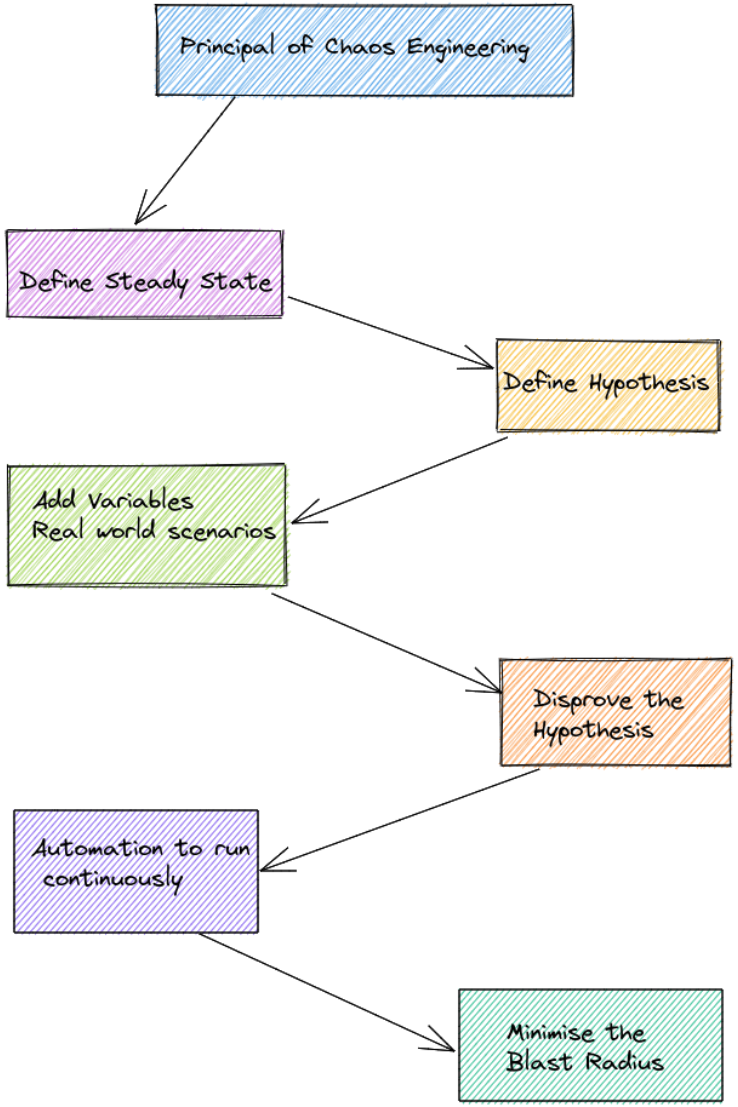
- Chaos Engineering is the discipline of experimenting with system in order to build confidence in the system's capability to withstand turbulent conditions in production.

- The objective of Chaos Engineering is to cause failure on purpose to identify where and under what conditions our system can fail, and improve our capacity.

- We intentionally inject fault in the system to check whether the system is resilient or not. The harder it is to disrupt the steady state, the more confidence we have in the behaviour of the system. If a weakness is uncovered, we now have a target for improvement before that behaviour manifests in the system at large.

Chaos In Practice

[Animation Link]



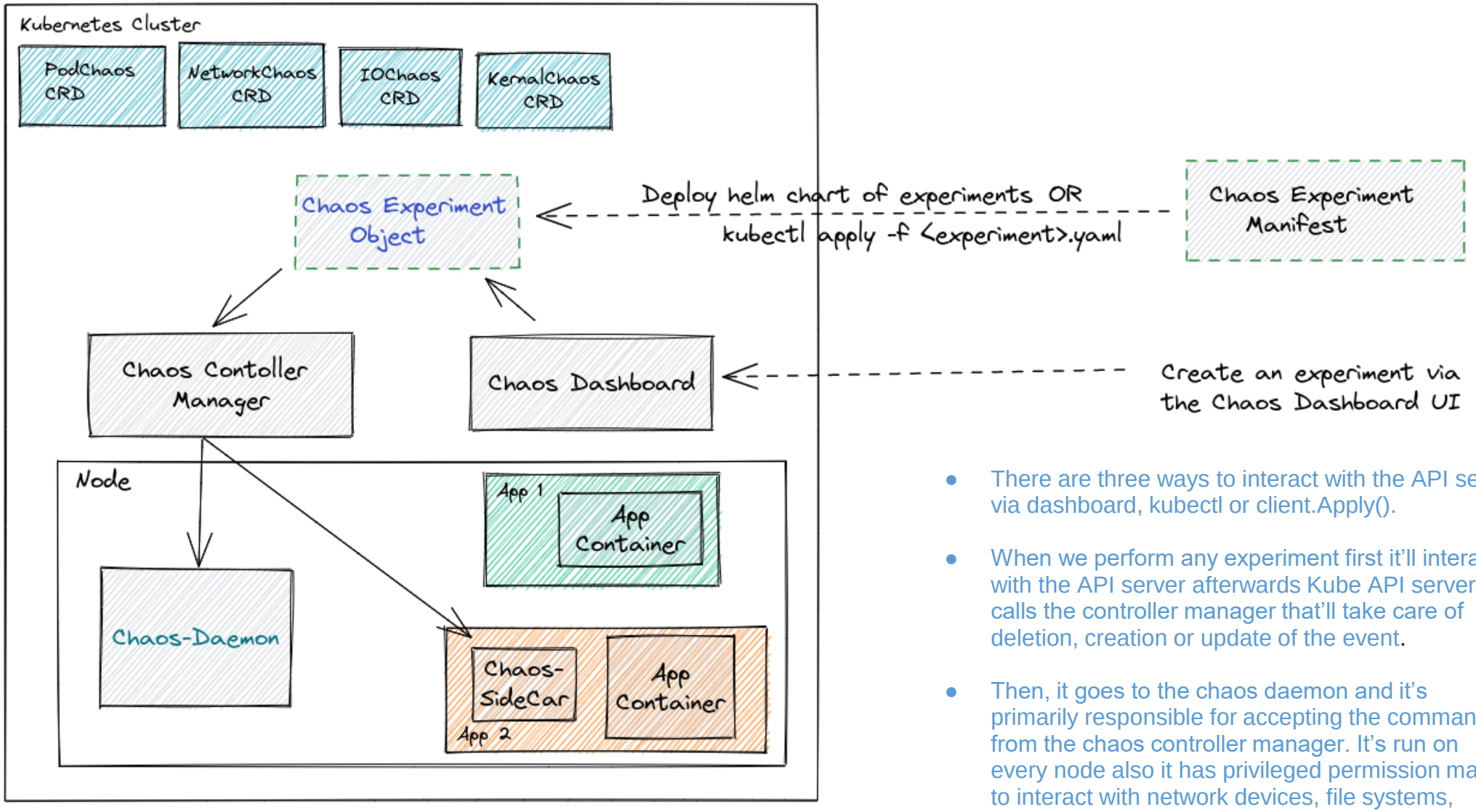
What is Chaos Mesh?

- Chaos Mesh is an open-source cloud-native Chaos Engineering platform that orchestrates chaos in Kubernetes environments.

- Chaos Mesh includes a fault injection method for complex systems on Kubernetes and covers faults in Pods, the network, the file system and even the kernel.

- Chaos Mesh is built on Kubernetes CRD (Custom Resource Definition). To manage different chaos experiments, Chaos Mesh defines multiple CRD types. These CRDs mainly categorized into three main fault types: basic resource faults, platform faults, and application-layer faults.

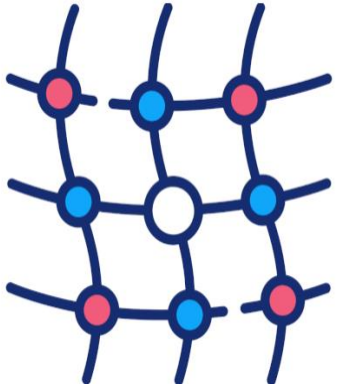
How Does It Work?



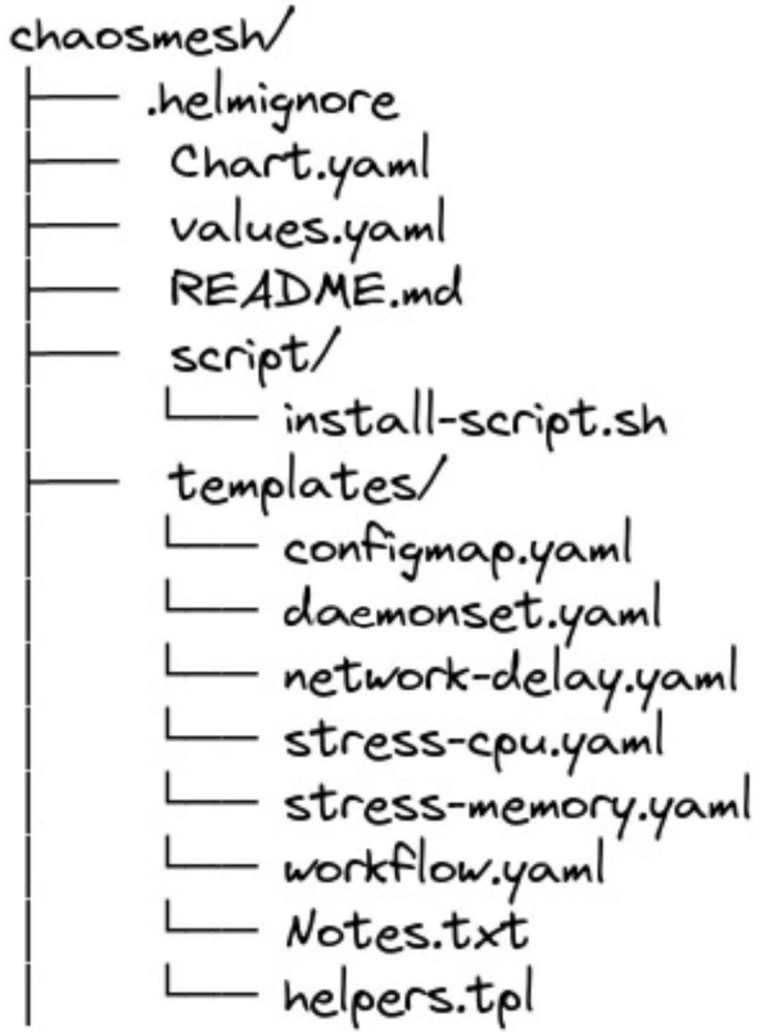
- There are three ways to interact with the API server via dashboard, kubectl or client.Apply().
- When we perform any experiment first it'll interact with the API server afterwards Kube API server calls the controller manager that'll take care of deletion, creation or update of the event.
- Then, it goes to the chaos daemon and it's primarily responsible for accepting the command from the chaos controller manager. It's run on every node also it has privileged permission mainly to interact with network devices, file systems, kernels by hacking into the target pod Namespace.

How We Implemented It?

+

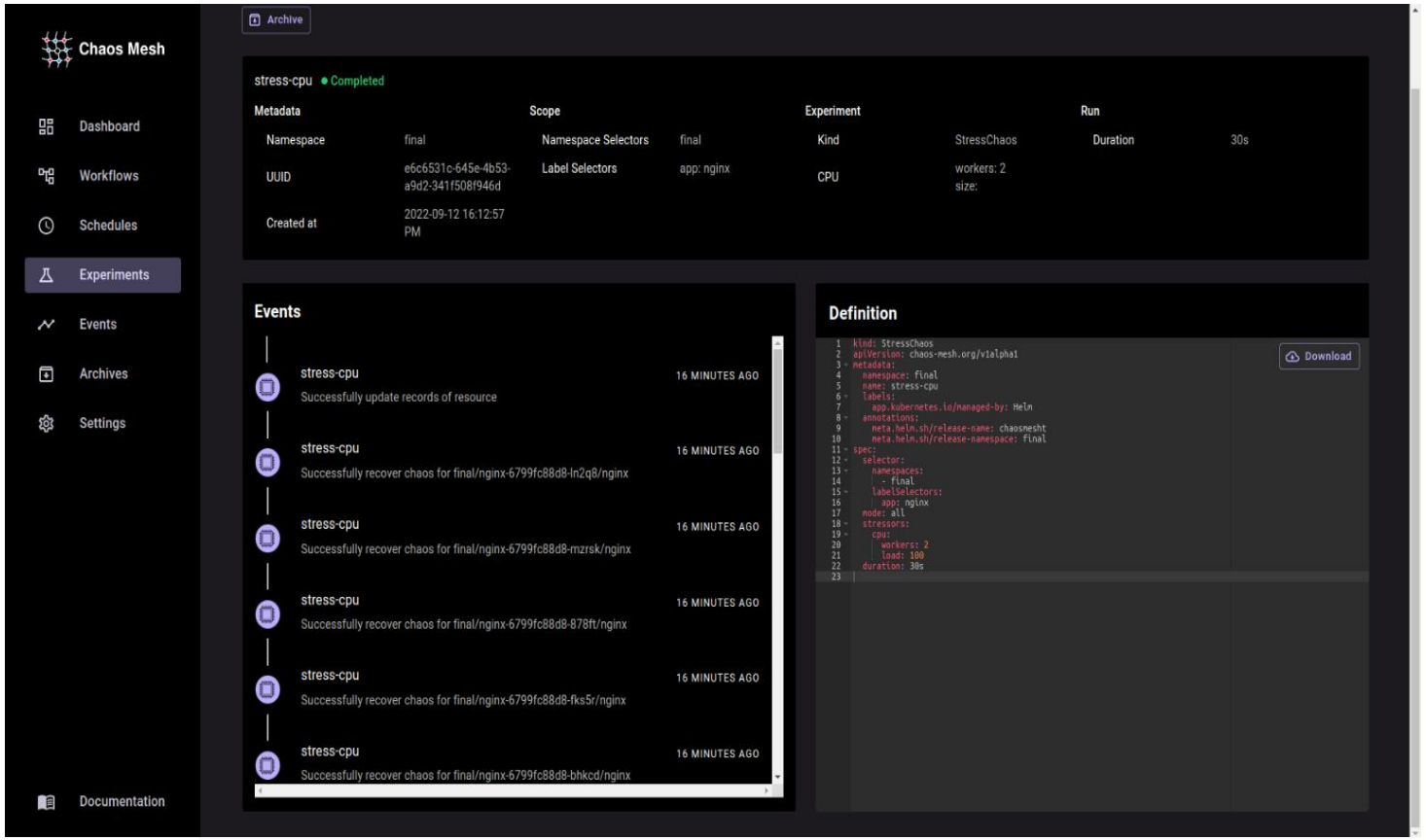


Helm Chart


chaosmesh/
.helmignore
Chart.yaml
values.yaml
README.md
script/
install-script.sh
templates/
configmap.yaml
daemonset.yaml
network-delay.yaml
stress-cpu.yaml
stress-memory.yaml
workflow.yaml
Notes.txt
helpers.tpl

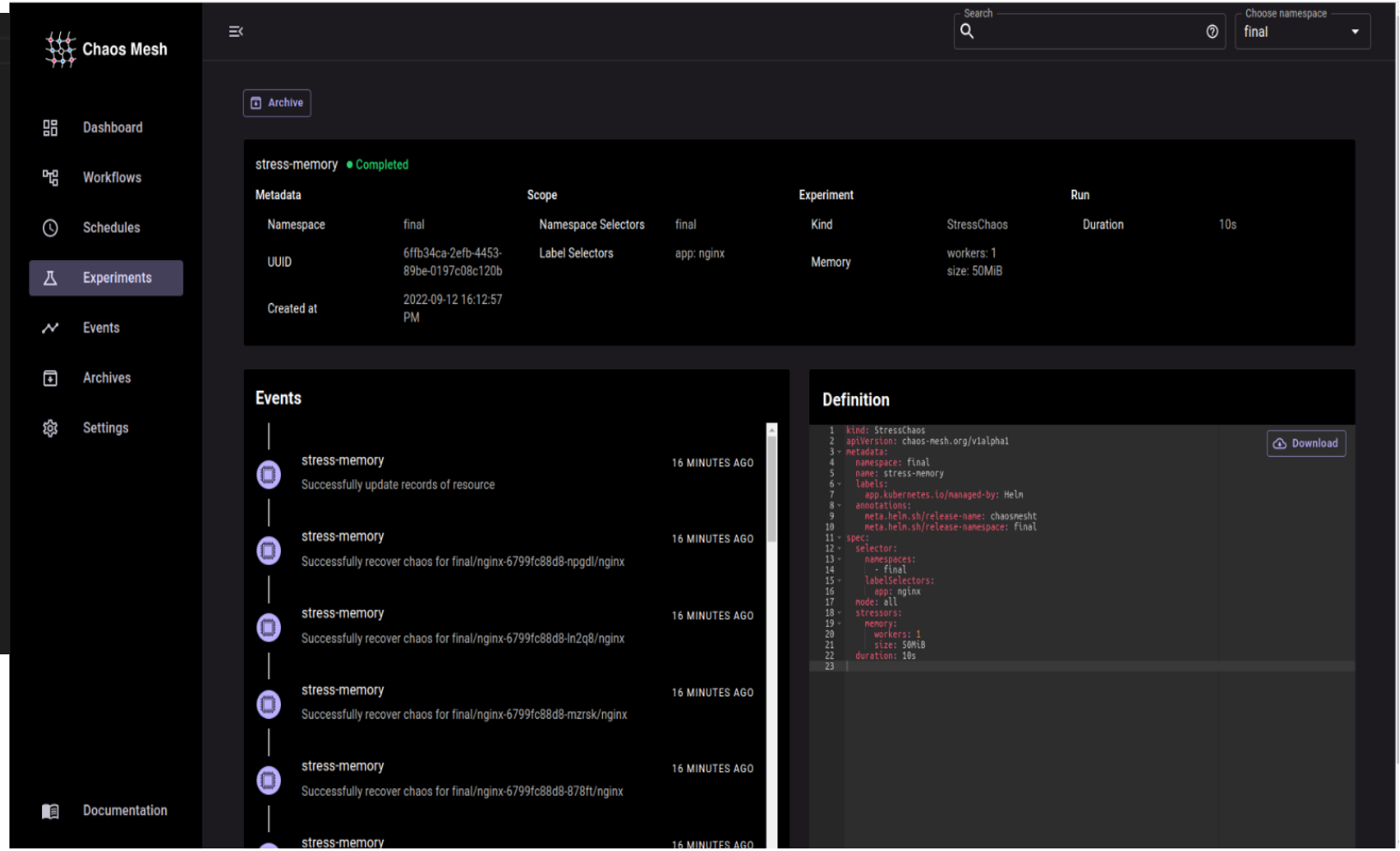
Stress CPU

```
{{- if .Values.experiments.stressCpu }}
{{- $namespaces := .Values.namespaces }}
{{- range $app := .Values.apps }}
apiVersion: chaos-mesh.org/v1alpha1
kind: StressChaos
metadata:
  name: stress-cpu
  namespace: {{ $.Release.Namespace }}
spec:
  mode: all
  selector:
    namespaces:
      - {{ $namespaces }}
    labelSelectors:
      app: "{{ $app }}"
  stressors:
    cpu:
      workers: 1
      load: 100
  duration: "30s"
{{- end }}
{{- end }}
```



Stress Memory

```
{{- if .Values.experiments.stressMemory }}
{{- $namespaces := .Values.namespaces }}
{{- range $app := .Values.apps }}
apiVersion: chaos-mesh.org/v1alpha1
kind: StressChaos
metadata:
  name: stress-memory
  namespace: {{ $.Release.Namespace }}
spec:
  mode: all
  selector:
    namespaces:
      - {{ $namespaces }}
    labelSelectors:
      app: "{{ $app }}"
  stressors:
    memory:
      workers: 1
      size: 50MiB
  duration: "10s"
{{- end }}
{{- end }}
```



Stress - Memory -CPU - Output

Before

```
Every 2.0s: kubectl top pods -n final

NAME                                      CPU(cores)   MEMORY(bytes)
chaos-controller-manager-867656dcf5-4mgc8   7m           35Mi
chaos-controller-manager-867656dcf5-lbwf2   1m           20Mi
chaos-controller-manager-867656dcf5-xkn2f   1m           18Mi
chaos-daemon-knzrm                          1m           18Mi
chaos-daemon-xm4fv                          1m           16Mi
chaos-dashboard-846949f9bb-5snw8            6m           45Mi
daemonset-jjthj                             0m           2Mi
daemonset-rxwdv                             0m           1Mi
nginx-6799fc88d8-7qb9x                      0m           2Mi
nginx-6799fc88d8-c2k7g                      0m           2Mi
nginx-6799fc88d8-fhhrt                      0m           2Mi
nginx-6799fc88d8-gtzwv                      0m           2Mi
nginx-6799fc88d8-lkxtb                      0m           2Mi
nginx-6799fc88d8-vgvjh                      0m           2Mi
nginx-6799fc88d8-wf7pg                      0m           2Mi
nginx-6799fc88d8-xhrs8                      0m           2Mi
```

After

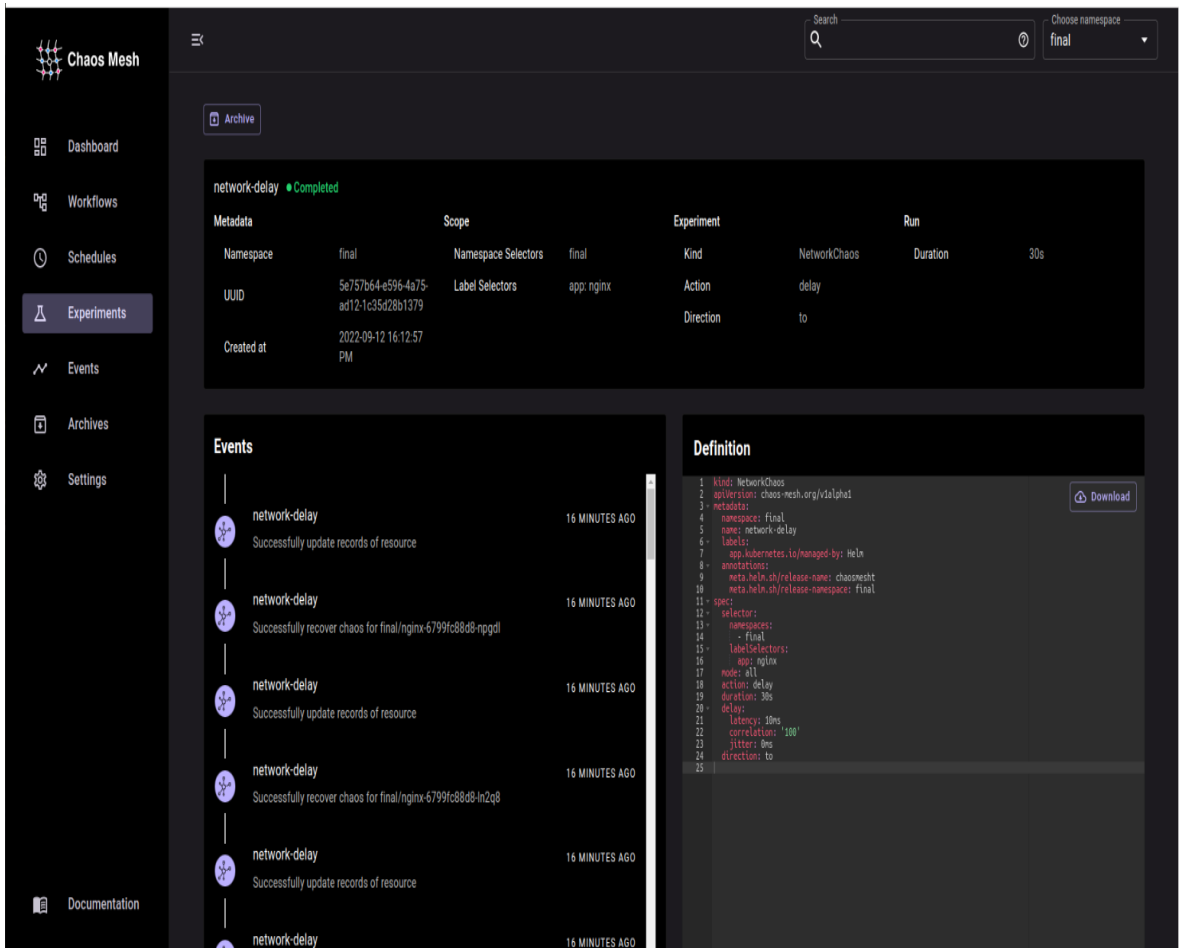
```
Every 2.0s: kubectl top pods -n final

NAME                                      CPU(cores)   MEMORY(bytes)
chaos-controller-manager-867656dcf5-4mgc8   7m           37Mi
chaos-controller-manager-867656dcf5-lbwf2   2m           20Mi
chaos-controller-manager-867656dcf5-xkn2f   1m           17Mi
chaos-daemon-knzrm                          24m          19Mi
chaos-daemon-xm4fv                          1m           18Mi
chaos-dashboard-846949f9bb-5snw8            7m           45Mi
daemonset-jjthj                             0m           2Mi
daemonset-rxwdv                             0m           1Mi
nginx-6799fc88d8-4sshm                      406m         6Mi
nginx-6799fc88d8-62j8n                      594m         6Mi
nginx-6799fc88d8-8rrlw                      510m         5Mi
nginx-6799fc88d8-c56p8                      356m         5Mi
nginx-6799fc88d8-mbm4b                      430m         6Mi
nginx-6799fc88d8-pnxhh                      503m         5Mi
nginx-6799fc88d8-rqqjr                      252m         2Mi
nginx-6799fc88d8-vvltm                      362m         5Mi
```

These workers will be scheduled and run for 30s in the pod, meaning we should expect to see our Nginx pods' CPU & Memory spike for 30s and then drop back to near 0.

Network Delay

That will introduce a latency of 10ms in the network of pods with labels app:nginx i.e nginx pod for the next 30 seconds.

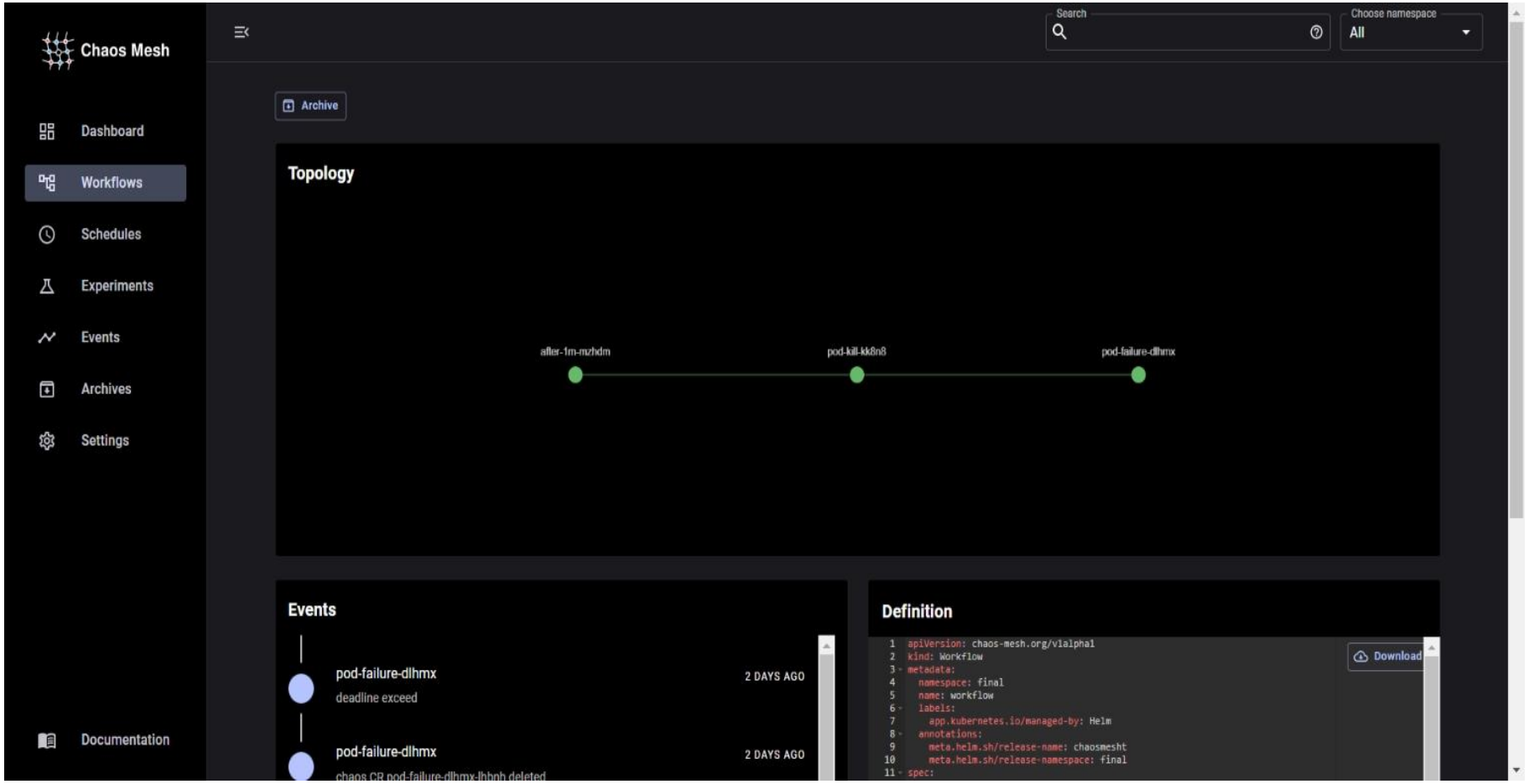
```yaml
{{- if .Values.experiments.networkDelay }}
{{- $namespaces := .Values.namespaces }}
{{- range $app := .Values.apps }}
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: network-delay
  namespace: {{ $.Release.Namespace }}
spec:
  action: delay
  mode: all
  duration: '30s'
  selector:
    namespaces:
      - {{ $namespaces }}
    labelSelectors:
      app: "{{ $app }}"
  delay:
    latency: '10ms' #indicates the network latency
    correlation: '100' #correlation b/w the current latency and previous one
    jitter: '0ms' #indicates the range of the network policy
  direction: to
{{- end }}
{{- end }}
```



Daemonset & ConfigMap

- To perform network chaos experiment it’s required to have NET_SCH_NETEM module installed on every node.
- Daemonset in Kubernetes allow you to run a pod on every node.
- And, we pack the script file in the ConfigMap to installed the network simulator on every node.

WorkFlow(Pod Kill & Pod Failure)



WorkFlow- Output

Before

```
NAME                                    READY   STATUS    RESTARTS   AGE
chaos-controller-manager-867656dcf5-4mgc8   1/1     Running   0          2d19h
chaos-controller-manager-867656dcf5-lbwf2   1/1     Running   0          2d19h
chaos-controller-manager-867656dcf5-xkn2f   1/1     Running   0          2d19h
chaos-daemon-knzrm                      1/1     Running   0          2d19h
chaos-daemon-xm4fv                      1/1     Running   0          2d19h
chaos-dashboard-846949f9bb-5snw8        1/1     Running   0          2d19h
daemonset-jjthj                         1/1     Running   0          2d19h
daemonset-rxwdv                         1/1     Running   0          2d19h
nginx-6799fc88d8-9vtcn                  1/1     Running   1          89s
nginx-6799fc88d8-cv7xx                  1/1     Running   1          89s
nginx-6799fc88d8-dmzhq                  1/1     Running   1          89s
nginx-6799fc88d8-gldwq                  1/1     Running   1          89s
nginx-6799fc88d8-j9nl2                  1/1     Running   1          89s
nginx-6799fc88d8-n5sdb                  1/1     Running   1          89s
nginx-6799fc88d8-rck4c                  1/1     Running   1          89s
nginx-6799fc88d8-xwzdr                  1/1     Running   1          89s
[nprasad@lxplus8s20 ~]$ kubectl get pods -n final
```

After

```
[nprasad@lxplus8s20 ~]$ kubectl get pods -n final
NAME                                    READY   STATUS    RESTARTS   AGE
chaos-controller-manager-867656dcf5-4mgc8   1/1     Running   0          2d19h
chaos-controller-manager-867656dcf5-lbwf2   1/1     Running   0          2d19h
chaos-controller-manager-867656dcf5-xkn2f   1/1     Running   0          2d19h
chaos-daemon-knzrm                      1/1     Running   0          2d19h
chaos-daemon-xm4fv                      1/1     Running   0          2d19h
chaos-dashboard-846949f9bb-5snw8        1/1     Running   0          2d19h
daemonset-jjthj                         1/1     Running   0          2d19h
daemonset-rxwdv                         1/1     Running   0          2d19h
nginx-6799fc88d8-9vtcn                  1/1     Running   2          2m46s
nginx-6799fc88d8-cv7xx                  1/1     Running   2          2m46s
nginx-6799fc88d8-dmzhq                  1/1     Running   2          2m46s
nginx-6799fc88d8-gldwq                  1/1     Running   2          2m46s
nginx-6799fc88d8-j9nl2                  1/1     Running   2          2m46s
nginx-6799fc88d8-n5sdb                  1/1     Running   2          2m46s
nginx-6799fc88d8-rck4c                  1/1     Running   2          2m46s
nginx-6799fc88d8-xwzdr                  1/1     Running   2          2m46s
[nprasad@lxplus8s20 ~]$ kubectl get pods -n final -All
```

Future Work

- Add platform faults, and application-layer faults.
- Integrate Chaos Mesh with other tools like Grafana, Gitlab.
- Provide more user-friendly features like notifications, scheduling, and visualization.

Thank You!

Questions?

nivedita.prasad@cern.ch

niveditaprasad81@gmail.com
: @NiveditaPrasa15