



## Your instructions for Chaos Engineering:

- The Chaos-Engineering experiment must be completed within 1 minute.
- When using k6 in steady-state definition, always select a request URL from the following options (other requests are invalid):
  1. <http://front-end.sock-shop.svc.cluster.local/>
  2. <http://front-end.sock-shop.svc.cluster.local/catalogue?size=10>
  3. <http://front-end.sock-shop.svc.cluster.local/detail.html?id=<ID>> Replace <ID> with an available ID: [ 03fef6ac-1896-4ce8-bd69-b798f85c6e0b , 3395a43e-2d88-40de-b95f-e00e1502085b , 510a0d7e-8e83-4193-b483-e27e09ddc34d , 808a2de1-1aaa-4c25-a9b9-6612e8f29a38 , 819e1fbf-8b7e-4f6d-811f-693534916a8b , 837ab141-399e-4c1f-9abc-bace40296bac , a0a4f044-b040-410d-8ead-4de0446aec7e , d3588630-ad8e-49df-bbd7-3167f7efb246 , zzz4f044-b040-410d-8ead-4de0446aec7e ]
  4. <http://front-end.sock-shop.svc.cluster.local/category/>
  5. <http://front-end.sock-shop.svc.cluster.local/category?tags=<TAG>> Replace <TAG> with an available tag: [ magic , action , blue , brown , black , sport , formal , red , green , skin , geek ]
  6. <http://front-end.sock-shop.svc.cluster.local/basket.html>



## Phase 0: Preprocessing

Cleaning the cluster `kind-chaos-eater-cluster` ... Done

```
$ kubectl delete workflow --all --context kind-chaos-eater-cluster -n chaos-  
workflow.chaos-mesh.org "chaos-experiment-20241127-041509" deleted  
$ kubectl delete workflownode --all --context kind-chaos-eater-cluster -n cl  
workflownode.chaos-mesh.org "fault-injection-phase-7xktg" deleted  
workflownode.chaos-mesh.org "fault-podchaos-7sc5f" deleted  
workflownode.chaos-mesh.org "fault-stresschaos-257q4" deleted  
workflownode.chaos-mesh.org "post-unittest-certs-dh-repl-logs-running-bd5ic"
```

Input instructions for your Chaos Engineering



```

workflownode.chaos-mesh.org "pre-unittest-carts-db-replicas-running-8h5z4" (
workflownode.chaos-mesh.org "pre-unittest-front-end-replica-running-njmf2" (
workflownode.chaos-mesh.org "pre-validation-parallel-workflows-499l9" delete
workflownode.chaos-mesh.org "pre-validation-phase-nxjfl" deleted
workflownode.chaos-mesh.org "the-entry-7kqk5" deleted
$ kubectl delete deployments --all --context kind-chaos-eater-cluster -n chaos
No resources found
$ kubectl delete pods --all --context kind-chaos-eater-cluster -n chaos-eater
pod "post-unittest-carts-db-replicas-running-bdjjc-8pfzb" deleted
pod "post-unittest-front-end-replica-running-h2fgg-fc7j8" deleted
pod "pre-unittest-carts-db-replicas-running-8h5z4-z22kb" deleted
pod "pre-unittest-front-end-replica-running-njmf2-pj82p" deleted
$ kubectl delete services --all --context kind-chaos-eater-cluster -n chaos
No resources found

```

```

$ kubectl delete all --all-namespaces --context kind-chaos-eater-cluster -l
pod "carts-75d9599844-5lf6h" deleted
pod "carts-75d9599844-nclz5" deleted
pod "carts-db-6794d4f987-4xnnc" deleted
pod "carts-db-6794d4f987-ldcv4" deleted
pod "catalogue-577bcf65d7-nksxg" deleted
pod "catalogue-577bcf65d7-rcbhd" deleted
pod "catalogue-db-56666f68cd-sdnjp" deleted
pod "catalogue-db-56666f68cd-w8s9n" deleted
pod "front-end-5cd74f858b-6rpnw" deleted
pod "front-end-5cd74f858b-r2774" deleted
pod "orders-77b6947c8d-f78s6" deleted
pod "orders-77b6947c8d-scm7g" deleted
pod "orders-db-59f7c5f79d-t92qn" deleted
pod "orders-db-59f7c5f79d-wxq5d" deleted
pod "payment-65b4dd6f9b-kzkql" deleted
pod "payment-65b4dd6f9b-vr7fp" deleted
pod "queue-master-778c957b8f-gpk9t" deleted
pod "queue-master-778c957b8f-gt97p" deleted
pod "rabbitmq-6db7899db6-657s6" deleted
pod "rabbitmq-6db7899db6-g2kdt" deleted
pod "session-db-7ff969cbdd-4bvrn" deleted
pod "session-db-7ff969cbdd-6jmcw" deleted
pod "shipping-5f477dd4bd-q8mr9" deleted
pod "shipping-5f477dd4bd-qz2kf" deleted
pod "user-86d655596-4xv2z" deleted
pod "user-86d655596-gtp85" deleted
pod "user-db-555dc88f6-24hbk" deleted
pod "user-db-555dc88f6-jcczn" deleted
service "carts" deleted

```

```
service "carts-db" deleted
service "catalogue" deleted
service "catalogue-db" deleted
service "front-end" deleted
service "orders" deleted
service "orders-db" deleted
service "payment" deleted
service "queue-master" deleted
service "rabbitmq" deleted
service "session-db" deleted
service "shipping" deleted
service "user" deleted
service "user-db" delete...ployment.apps "queue-master" deleted
deployment.apps "rabbitmq" deleted
deployment.apps "session-db" deleted
deployment.apps "shipping" deleted
deployment.apps "user" deleted
deployment.apps "user-db" deleted
replicaset.apps "carts-75d9599844" deleted
replicaset.apps "carts-7b44c6f9f" deleted
replicaset.apps "carts-db-6794d4f987" deleted
replicaset.apps "carts-db-75b7fbdbbb" deleted
replicaset.apps "catalogue-577bcf65d7" deleted
replicaset.apps "catalogue-594894b47b" deleted
replicaset.apps "catalogue-db-56666f68cd" deleted
replicaset.apps "catalogue-db-68d6947649" deleted
replicaset.apps "front-end-5cd74f858b" deleted
replicaset.apps "front-end-75cb4d77df" deleted
replicaset.apps "orders-77b6947c8d" deleted
replicaset.apps "orders-8575b8c969" deleted
replicaset.apps "orders-db-59f7c5f79d" deleted
replicaset.apps "orders-db-76dfdc6865" deleted
replicaset.apps "payment-65b4dd6f9b" deleted
replicaset.apps "payment-7855f7898c" deleted
replicaset.apps "queue-master-584db96944" deleted
replicaset.apps "queue-master-778c957b8f" deleted
replicaset.apps "rabbitmq-6db7899db6" deleted
replicaset.apps "rabbitmq-7f84854df5" deleted
replicaset.apps "session-db-6fc87fc86c" deleted
replicaset.apps "session-db-7ff969cbdd" deleted
replicaset.apps "shipping-5f477dd4bd" deleted
replicaset.apps "shipping-77669895df" deleted
replicaset.apps "user-86bfdd744f" deleted
replicaset.apps "user-86d655596" deleted
replicaset.apps "user-db-555dc88f6" deleted
replicaset.apps "user-db-85c4bb8bd7" deleted
```

## K8s manifest(s) to be deployed:

sock-shop-2/manifests/00-sock-shop-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: sock-shop
```

sock-shop-2/manifests/01-carts-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: carts
  labels:
    name: carts
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: carts
  template:
    metadata:
      labels:
        name: carts
    spec:
      containers:
        - name: carts
          image: weaveworksdemos/carts:0.4.8
          env:
            - name: JAVA_OPTS
              value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/./urandom
          resources:
            limits:
              cpu: 300m
              memory: 500Mi
            requests:
              cpu: 100m
              memory: 200Mi
      ports:
        - containerPort: 80
      securityContext:
        runAsNonRoot: true
        runAsUser: 10001
```

```

    capabilities:
      drop:
        - all
      add:
        - NET_BIND_SERVICE
    readOnlyRootFilesystem: true
  volumeMounts:
    - mountPath: /tmp
      name: tmp-volume
  volumes:
    - name: tmp-volume
      emptyDir:
        medium: Memory
  nodeSelector:
    beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/02-carts-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: carts
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: carts
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80
  selector:
    name: carts

```

sock-shop-2/manifests/03-carts-db-dep.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: carts-db
  labels:
    name: carts-db
  namespace: sock-shop
spec:

```

```

replicas: 2
selector:
  matchLabels:
    name: carts-db
template:
  metadata:
    labels:
      name: carts-db
  spec:
    containers:
      - name: carts-db
        image: mongo
        ports:
          - name: mongo
            containerPort: 27017
        securityContext:
          capabilities:
            drop:
              - all
            add:
              - CHOWN
              - SETGID
              - SETUID
          readOnlyRootFilesystem: true
        volumeMounts:
          - mountPath: /tmp
            name: tmp-volume
    volumes:
      - name: tmp-volume
        emptyDir:
          medium: Memory
    nodeSelector:
      beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/04-carts-db-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: carts-db
  labels:
    name: carts-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on

```

```
- port: 27017
  targetPort: 27017
selector:
  name: carts-db
```

sock-shop-2/manifests/05-catalogue-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: catalogue
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: catalogue
  template:
    metadata:
      labels:
        name: catalogue
    spec:
      containers:
        - name: catalogue
          image: weaveworksdemos/catalogue:0.3.5
          command: ["/app"]
          args:
            - -port=80
          resources:
            limits:
              cpu: 200m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 100Mi
          ports:
            - containerPort: 80
      securityContext:
        runAsNonRoot: true
        runAsUser: 10001
        capabilities:
          drop:
            - all
          add:
```

```

      - NET_BIND_SERVICE
      readOnlyRootFilesystem: true
    livenessProbe:
      httpGet:
        path: /health
        port: 80
      initialDelaySeconds: 300
      periodSeconds: 3
    readinessProbe:
      httpGet:
        path: /health
        port: 80
      initialDelaySeconds: 180
      periodSeconds: 3
    nodeSelector:
      beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/06-catalogue-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: catalogue
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: catalogue
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80
  selector:
    name: catalogue

```

sock-shop-2/manifests/07-catalogue-db-dep.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: catalogue-db
  labels:
    name: catalogue-db
  namespace: sock-shop

```

```

spec:
  replicas: 2
  selector:
    matchLabels:
      name: catalogue-db
  template:
    metadata:
      labels:
        name: catalogue-db
    spec:
      containers:
        - name: catalogue-db
          image: weaveworksdemos/catalogue-db:0.3.0
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: fake_password
            - name: MYSQL_DATABASE
              value: socksdb
          ports:
            - name: mysql
              containerPort: 3306
      nodeSelector:
        beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/08-catalogue-db-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: catalogue-db
  labels:
    name: catalogue-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 3306
      targetPort: 3306
  selector:
    name: catalogue-db

```

sock-shop-2/manifests/09-front-end-dep.yaml

```

apiVersion: apps/v1
kind: Deployment

```

```
metadata:
  name: front-end
  namespace: sock-shop
spec:
  replicas: 1
  selector:
    matchLabels:
      name: front-end
  template:
    metadata:
      labels:
        name: front-end
    spec:
      containers:
        - name: front-end
          image: weaveworksdemos/front-end:0.3.12
          resources:
            limits:
              cpu: 300m
              memory: 1000Mi
            requests:
              cpu: 100m
              memory: 300Mi
          ports:
            - containerPort: 8079
          env:
            - name: SESSION_REDIS
              value: "true"
          securityContext:
            runAsNonRoot: true
            runAsUser: 10001
            capabilities:
              drop:
                - all
            readOnlyRootFilesystem: true
      livenessProbe:
        httpGet:
          path: /
          port: 8079
        initialDelaySeconds: 300
        periodSeconds: 3
      readinessProbe:
        httpGet:
          path: /
          port: 8079
        initialDelaySeconds: 30
        periodSeconds: 3
```

```
nodeSelector:  
  beta.kubernetes.io/os: linux
```

sock-shop-2/manifests/10-front-end-svc.yaml

```
apiVersion: v1  
kind: Service  
metadata:  
  name: front-end  
  annotations:  
    prometheus.io/scrape: 'true'  
  labels:  
    name: front-end  
  namespace: sock-shop  
spec:  
  type: NodePort  
  ports:  
    - port: 80  
      targetPort: 8079  
      nodePort: 30001  
  selector:  
    name: front-end
```

sock-shop-2/manifests/11-orders-dep.yaml

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: orders  
  labels:  
    name: orders  
  namespace: sock-shop  
spec:  
  replicas: 2  
  selector:  
    matchLabels:  
      name: orders  
  template:  
    metadata:  
      labels:  
        name: orders  
    spec:  
      containers:  
        - name: orders  
          image: weaveworksdemos/orders:0.4.7
```

```

env:
  - name: JAVA_OPTS
    value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/./urandom
resources:
  limits:
    cpu: 500m
    memory: 500Mi
  requests:
    cpu: 100m
    memory: 300Mi
ports:
  - containerPort: 80
securityContext:
  runAsNonRoot: true
  runAsUser: 10001
capabilities:
  drop:
    - all
  add:
    - NET_BIND_SERVICE
  readOnlyRootFilesystem: true
volumeMounts:
  - mountPath: /tmp
    name: tmp-volume
volumes:
  - name: tmp-volume
    emptyDir:
      medium: Memory
nodeSelector:
  beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/12-orders-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: orders
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: orders
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80

```

```
targetPort: 80
selector:
  name: orders
```

sock-shop-2/manifests/13-orders-db-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-db
  labels:
    name: orders-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: orders-db
  template:
    metadata:
      labels:
        name: orders-db
    spec:
      containers:
        - name: orders-db
          image: mongo
          ports:
            - name: mongo
              containerPort: 27017
          securityContext:
            capabilities:
              drop:
                - all
              add:
                - CHOWN
                - SETGID
                - SETUID
            readOnlyRootFilesystem: true
          volumeMounts:
            - mountPath: /tmp
              name: tmp-volume
      volumes:
        - name: tmp-volume
          emptyDir:
            medium: Memory
```

```
nodeSelector:
  beta.kubernetes.io/os: linux
```

sock-shop-2/manifests/14-orders-db-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: orders-db
  labels:
    name: orders-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 27017
      targetPort: 27017
  selector:
    name: orders-db
```

sock-shop-2/manifests/15-payment-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payment
  labels:
    name: payment
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: payment
  template:
    metadata:
      labels:
        name: payment
    spec:
      containers:
        - name: payment
          image: weaveworksdemos/payment:0.4.3
          resources:
            limits:
              cpu: 200m
```

```

        memory: 200Mi
      requests:
        cpu: 99m
        memory: 100Mi
    ports:
      - containerPort: 80
    securityContext:
      runAsNonRoot: true
      runAsUser: 10001
    capabilities:
      drop:
        - all
      add:
        - NET_BIND_SERVICE
    readOnlyRootFilesystem: true
  livenessProbe:
    httpGet:
      path: /health
      port: 80
    initialDelaySeconds: 300
    periodSeconds: 3
  readinessProbe:
    httpGet:
      path: /health
      port: 80
    initialDelaySeconds: 180
    periodSeconds: 3
  nodeSelector:
    beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/16-payment-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: payment
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: payment
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80

```

```
selector:
  name: payment
```

sock-shop-2/manifests/17-queue-master-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: queue-master
  labels:
    name: queue-master
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: queue-master
  template:
    metadata:
      labels:
        name: queue-master
    spec:
      containers:
        - name: queue-master
          image: weaveworksdemos/queue-master:0.3.1
          env:
            - name: JAVA_OPTS
              value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/./urandom
          resources:
            limits:
              cpu: 300m
              memory: 500Mi
            requests:
              cpu: 100m
              memory: 300Mi
          ports:
            - containerPort: 80
      nodeSelector:
        beta.kubernetes.io/os: linux
```

sock-shop-2/manifests/18-queue-master-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
```

```

name: queue-master
annotations:
  prometheus.io/scrape: 'true'
labels:
  name: queue-master
namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80
  selector:
    name: queue-master

```

sock-shop-2/manifests/19-rabbitmq-dep.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
  labels:
    name: rabbitmq
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: rabbitmq
  template:
    metadata:
      labels:
        name: rabbitmq
      annotations:
        prometheus.io/scrape: "false"
    spec:
      containers:
        - name: rabbitmq
          image: rabbitmq:3.6.8-management
          ports:
            - containerPort: 15672
              name: management
            - containerPort: 5672
              name: rabbitmq
      securityContext:
        capabilities:
          drop:

```

```

      - all
    add:
      - CHOWN
      - SETGID
      - SETUID
      - DAC_OVERRIDE
    readOnlyRootFilesystem: true
  - name: rabbitmq-exporter
    image: kbudde/rabbitmq-exporter
    ports:
      - containerPort: 9090
        name: exporter
    nodeSelector:
      beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/20-rabbitmq-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: rabbitmq
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/port: '9090'
  labels:
    name: rabbitmq
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 5672
      name: rabbitmq
      targetPort: 5672
    - port: 9090
      name: exporter
      targetPort: exporter
      protocol: TCP
  selector:
    name: rabbitmq

```

sock-shop-2/manifests/21-session-db-dep.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:

```

```

name: session-db
labels:
  name: session-db
namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: session-db
  template:
    metadata:
      labels:
        name: session-db
      annotations:
        prometheus.io.scrape: "false"
    spec:
      containers:
        - name: session-db
          image: redis:alpine
          ports:
            - name: redis
              containerPort: 6379
          securityContext:
            capabilities:
              drop:
                - all
              add:
                - CHOWN
                - SETGID
                - SETUID
            readOnlyRootFilesystem: true
      nodeSelector:
        beta.kubernetes.io/os: linux

```

`sock-shop-2/manifests/22-session-db-svc.yaml`

```

apiVersion: v1
kind: Service
metadata:
  name: session-db
  labels:
    name: session-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on

```

```
- port: 6379
  targetPort: 6379
selector:
  name: session-db
```

sock-shop-2/manifests/23-shipping-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: shipping
  labels:
    name: shipping
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: shipping
  template:
    metadata:
      labels:
        name: shipping
    spec:
      containers:
        - name: shipping
          image: weaveworksdemos/shipping:0.4.8
          env:
            - name: ZIPKIN
              value: zipkin.jaeger.svc.cluster.local
            - name: JAVA_OPTS
              value: -Xms64m -Xmx128m -XX:+UseG1GC -Djava.security.egd=file:/dev/./urandom
          resources:
            limits:
              cpu: 300m
              memory: 500Mi
            requests:
              cpu: 100m
              memory: 300Mi
          ports:
            - containerPort: 80
      securityContext:
        runAsNonRoot: true
        runAsUser: 10001
        capabilities:
          drop:
```

```

      - all
    add:
      - NET_BIND_SERVICE
    readOnlyRootFilesystem: true
    volumeMounts:
      - mountPath: /tmp
        name: tmp-volume
    volumes:
      - name: tmp-volume
        emptyDir:
          medium: Memory
    nodeSelector:
      beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/24-shipping-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: shipping
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: shipping
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80
  selector:
    name: shipping

```

sock-shop-2/manifests/25-user-dep.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: user
  labels:
    name: user
  namespace: sock-shop
spec:
  replicas: 2
  selector:

```

```

matchLabels:
  name: user
template:
  metadata:
    labels:
      name: user
  spec:
    containers:
      - name: user
        image: weaveworksdemos/user:0.4.7
        resources:
          limits:
            cpu: 300m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
          - containerPort: 80
        env:
          - name: mongo
            value: user-db:27017
        securityContext:
          runAsNonRoot: true
          runAsUser: 10001
          capabilities:
            drop:
              - all
            add:
              - NET_BIND_SERVICE
          readOnlyRootFilesystem: true
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 300
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 180
          periodSeconds: 3
    nodeSelector:
      beta.kubernetes.io/os: linux

```

```
apiVersion: v1
kind: Service
metadata:
  name: user
  annotations:
    prometheus.io/scrape: 'true'
  labels:
    name: user
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 80
      targetPort: 80
  selector:
    name: user
```

sock-shop-2/manifests/27-user-db-dep.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-db
  labels:
    name: user-db
  namespace: sock-shop
spec:
  replicas: 2
  selector:
    matchLabels:
      name: user-db
  template:
    metadata:
      labels:
        name: user-db
    spec:
      containers:
        - name: user-db
          image: weaveworksdemos/user-db:0.3.0

      ports:
        - name: mongo
          containerPort: 27017
      securityContext:
        capabilities:
```

```

      drop:
        - all
      add:
        - CHOWN
        - SETGID
        - SETUID
      readOnlyRootFilesystem: true
    volumeMounts:
      - mountPath: /tmp
        name: tmp-volume
    volumes:
      - name: tmp-volume
        emptyDir:
          medium: Memory
    nodeSelector:
      beta.kubernetes.io/os: linux

```

sock-shop-2/manifests/28-user-db-svc.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: user-db
  labels:
    name: user-db
  namespace: sock-shop
spec:
  ports:
    # the port that this service should serve on
    - port: 27017
      targetPort: 27017
  selector:
    name: user-db

```

## Deploying resources... Done

```

$ skaffold run --kube-context kind-chaos-eater-cluster -l project=chaos-eater
No tags generated
Starting test...
Starting deploy...
Loading images into kind cluster nodes...
Images loaded in 80ns
- namespace/sock-shop unchanged
- Warning: spec.template.spec.nodeSelector[beta.kubernetes.io/os]: deprecated
- deployment.apps/carts created

```

- service/carts created
- deployment.apps/carts-db created
- service/carts-db created
- deployment.apps/catalogue created
- service/catalogue created
- deployment.apps/catalogue-db created
- service/catalogue-db created
- deployment.apps/front-end created
- service/front-end created
- deployment.apps/orders created
- service/orders created
- deployment.apps/orders-db created
- service/orders-db created
- deployment.apps/payment created
- service/payment created
- deployment.apps/queue-master created
- service/queue-master created
- deployment.apps/rabbitmq created
- service/rabbitmq created
- deployment.apps/session-db created
- service/session-db created
- deployment.apps/shipping created
- service/shipping created
- deployment.apps/user created
- service/user created
- deployment.apps/user-db created
- service/user-db created

Waiting for deployments to stabilize...

- sock-shop:deployment/carts is ready. [13/14 deployment(s) still pending]
- sock-shop:deployment/carts-db: creating container carts-db
  - sock-shop:pod/carts-db-84dd74485f-jn289: creating container carts-db
  - sock-shop:pod/carts-db-84dd74485f-pggt5: creating container carts...p
- sock-shop:deployment/orders: waiting for rollout to finish: 1 of 2 updated
- sock-shop:deployment/payment: creating container payment
  - sock-shop:pod/payment-6b5cf84897-x5phb: creating container payment
  - sock-shop:pod/payment-6b5cf84897-mn7d6: creating container payment
- sock-shop:deployment/shipping: creating container shipping
  - sock-shop:pod/shipping-6cc64f8975-cl4ql: creating container shipping
  - sock-shop:pod/shipping-6cc64f8975-zz945: creating container shipping
- sock-shop:deployment/user: creating container user
  - sock-shop:pod/user-84fb6b864c-lptjk: creating container user
- sock-shop:deployment/orders is ready. [7/14 deployment(s) still pending]
- sock-shop:deployment/rabbitmq is ready. [6/14 deployment(s) still pending]
- sock-shop:deployment/shipping is ready. [5/14 deployment(s) still pending]
- sock-shop:deployment/orders-db is ready. [4/14 deployment(s) still pending]
- sock-shop:deployment/user: waiting for rollout to finish: 0 of 2 updated
- sock-shop:deployment/front-end is ready. [3/14 deployment(s) still pending]

- sock-shop:deployment/payment is ready. [2/14 deployment(s) still pending]
- sock-shop:deployment/catalogue is ready. [1/14 deployment(s) still pending]
- sock-shop:deployment/user is ready.

Deployments stabilized in 3 minutes 5.307 seconds

You can also run [skaffold run --tail] to get the logs

## Resource statuses

```
$ kubectl get all --all-namespaces --context kind-chaos-eater-cluster --selector=sock-shop
```

NAMESPACE	NAME	READY	STATUS	RESTARTS
sock-shop	pod/carts-56cc746557-d4jm9	1/1	Running	0
sock-shop	pod/carts-56cc746557-dvqt5	1/1	Running	0
sock-shop	pod/carts-db-84dd74485f-jn289	1/1	Running	0
sock-shop	pod/carts-db-84dd74485f-pggt5	1/1	Running	0
sock-shop	pod/catalogue-8695b4dcfd-tldhj	1/1	Running	0
sock-shop	pod/catalogue-8695b4dcfd-vbml7	1/1	Running	0
sock-shop	pod/catalogue-db-68bb48f867-pbnwt	1/1	Running	0
sock-shop	pod/catalogue-db-68bb48f867-rg594	1/1	Running	0
sock-shop	pod/front-end-c669bb67f-l4fvb	1/1	Running	0
sock-shop	pod/orders-697c586dd7-952x5	1/1	Running	0
sock-shop	pod/orders-697c586dd7-npwkz	1/1	Running	0
sock-shop	pod/orders-db-694d59df67-f972v	1/1	Running	0
sock-shop	pod/orders-db-694d59df67-hc2tj	1/1	Running	0
sock-shop	pod/payment-6b5cf84897-mn7d6	1/1	Running	0
sock-shop	pod/payment-6b5cf84897-x5phb	1/1	Running	0
sock-shop	pod/queue-master-85c79fbdf8-7h1bz	1/1	Running	0
sock-shop	pod/queue-master-85c79fbdf8-pnlw8	1/1	Running	0
sock-shop	pod/rabbitmq-c5b8d94c7	2	3m8s	
sock-shop	deployment.apps/user	2/2	2	2
sock-shop	deployment.apps/user-db	2/2	2	2

NAMESPACE	NAME	DESIRED	CURRENT	RESTARTS
sock-shop	replicaset.apps/carts-56cc746557	2	2	2
sock-shop	replicaset.apps/carts-db-84dd74485f	2	2	2
sock-shop	replicaset.apps/catalogue-8695b4dcfd	2	2	2
sock-shop	replicaset.apps/catalogue-db-68bb48f867	2	2	2
sock-shop	replicaset.apps/front-end-c669bb67f	1	1	1
sock-shop	replicaset.apps/orders-697c586dd7	2	2	2
sock-shop	replicaset.apps/orders-db-694d59df67	2	2	2
sock-shop	replicaset.apps/payment-6b5cf84897	2	2	2
sock-shop	replicaset.apps/queue-master-85c79fbdf8	2	2	2
sock-shop	replicaset.apps/rabbitmq-c5b8d94c7	2	2	2
sock-shop	replicaset.apps/session-db-65c8df9f69	2	2	2
sock-shop	replicaset.apps/shipping-6cc64f8975	2	2	2

sock-shop	replicaset.apps/user-84fb6b864c	2	2	2
sock-shop	replicaset.apps/user-db-745c77dd65	2	2	2

## Summary of each manifest:

`sock-shop-2/manifests/00-sock-shop-ns.yaml`

- This manifest defines a Kubernetes Namespace.
- The Namespace is named 'sock-shop'.
- Namespaces are used to organize and manage resources within a Kubernetes cluster, providing a way to divide cluster resources between multiple users or teams.

`sock-shop-2/manifests/01-carts-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'carts' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'carts' application running.
- The Deployment uses the Docker image 'weaveworksdemos/carts:0.4.8'.
- Environment variables are set for Java options to optimize memory usage and disable certain features.
- Resource limits and requests are defined, with a maximum of 300m CPU and 500Mi memory, and a minimum of 100m CPU and 200Mi memory.
- The application listens on port 80 within the container.
- Security settings ensure the container runs as a non-root user with specific capabilities and a read-only root filesystem.
- A temporary volume is mounted at '/tmp' using an in-memory empty directory.
- The Deployment is scheduled to run on nodes with the Linux operating system.

`sock-shop-2/manifests/02-carts-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'carts'.
- It is annotated for Prometheus scraping with 'prometheus.io/scrape: true'.
- The Service is labeled with 'name: carts'.
- It is deployed in the 'sock-shop' namespace.
- The Service exposes port 80 and directs traffic to the same port on the target pods.
- It selects pods with the label 'name: carts' to route traffic to.

`sock-shop-2/manifests/03-carts-db-dep.yaml`

- This manifest defines a Deployment in Kubernetes.

- The Deployment is named 'carts-db' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'carts-db' pod running.
- The pods are selected based on the label 'name: carts-db'.
- Each pod runs a single container using the 'mongo' image.
- The container exposes port 27017, which is the default port for MongoDB.
- Security settings are applied to drop all capabilities and only add CHOWN, SETGID, and SETUID.
- The root filesystem of the container is set to read-only for security purposes.
- A temporary volume is mounted at '/tmp' using an in-memory emptyDir volume.
- The pods are scheduled to run on nodes with the operating system labeled as 'linux'.

`sock-shop-2/manifests/04-carts-db-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'carts-db'.
- It is labeled with 'name: carts-db'.
- The Service is created in the 'sock-shop' namespace.
- It exposes port 27017 and directs traffic to the same port on the target pods.
- The Service selects pods with the label 'name: carts-db' to route traffic to them.

`sock-shop-2/manifests/05-catalogue-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'catalogue' and is part of the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'catalogue' application running.
- The Deployment uses the Docker image 'weaveworksdemos/catalogue:0.3.5'.
- The application runs with the command '/app' and listens on port 80.
- Resource limits are set to 200m CPU and 200Mi memory, with requests for 100m CPU and 100Mi memory.
- The container is configured to run as a non-root user with user ID 10001.
- Security settings include dropping all capabilities except 'NET\_BIND\_SERVICE' and using a read-only root filesystem.
- Liveness and readiness probes are configured to check the '/health' endpoint on port 80, with initial delays of 300 and 180 seconds respectively.
- The Deployment is scheduled to run on nodes with the Linux operating system.

`sock-shop-2/manifests/06-catalogue-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'catalogue'.

- It is annotated to enable Prometheus scraping with 'prometheus.io/scrape: true'.
- The Service is labeled with 'name: catalogue'.
- It is deployed in the 'sock-shop' namespace.
- The Service exposes port 80 and directs traffic to the same port on the selected pods.
- It selects pods with the label 'name: catalogue' to route traffic to.

`sock-shop-2/manifests/07-catalogue-db-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'catalogue-db'.
- It is located in the 'sock-shop' namespace.
- The Deployment will create 2 replicas of the pod.
- Each pod will run a container from the image 'weaveworksdemos/catalogue-db:0.3.0'.
- The container is configured with environment variables for 'MYSQL\_ROOT\_PASSWORD' and 'MYSQL\_DATABASE'.
- The container exposes port 3306, which is typically used for MySQL.
- The pods are scheduled to run on nodes with the operating system labeled as Linux.

`sock-shop-2/manifests/08-catalogue-db-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'catalogue-db'.
- It is located in the 'sock-shop' namespace.
- The Service is configured to expose port 3306.
- It targets the same port (3306) on the pods.
- The Service selects pods with the label 'name: catalogue-db'.

`sock-shop-2/manifests/09-front-end-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'front-end' and is located in the 'sock-shop' namespace.
- It specifies that there should be 1 replica of the front-end application running.
- The Deployment uses a selector to match pods with the label 'name: front-end'.
- The pod template includes a single container named 'front-end'.
- The container uses the image 'weaveworksdemos/front-end:0.3.12'.
- Resource limits are set for the container: 300m CPU and 1000Mi memory.
- Resource requests are set for the container: 100m CPU and 300Mi memory.
- The container exposes port 8079.
- An environment variable 'SESSION\_REDIS' is set to 'true'.
- Security context is configured to run the container as a non-root user with user ID 10001.

- All additional Linux capabilities are dropped, and the root filesystem is set to read-only.
- A liveness probe is configured to check the '/' path on port 8079, with an initial delay of 300 seconds and a period of 3 seconds.
- A readiness probe is also configured to check the '/' path on port 8079, with an initial delay of 30 seconds and a period of 3 seconds.
- The node selector ensures that the pod runs on nodes with the operating system labeled as Linux.

`sock-shop-2/manifests/10-front-end-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'front-end'.
- It is located in the 'sock-shop' namespace.
- The Service type is 'NodePort', which exposes the service on each Node's IP at a static port.
- It listens on port 80 and forwards traffic to target port 8079 on the pods.
- The nodePort is set to 30001, which is the port on each node where the service can be accessed externally.
- The Service is configured to be scraped by Prometheus for monitoring, as indicated by the annotation 'prometheus.io/scrape: true'.
- It selects pods with the label 'name: front-end' to route traffic to.

`sock-shop-2/manifests/11-orders-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'orders' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'orders' application running.
- The Deployment uses the 'weaveworksdemos/orders:0.4.7' Docker image for the container.
- Environment variables are set for Java options to optimize memory usage and disable certain features.
- Resource limits and requests are defined, with a maximum of 500m CPU and 500Mi memory, and a minimum of 100m CPU and 300Mi memory.
- The container listens on port 80.
- Security settings ensure the container runs as a non-root user with specific capabilities and a read-only root filesystem.
- A temporary volume is mounted at '/tmp' using an in-memory empty directory.
- The Deployment is scheduled to run on nodes with the Linux operating system.

`sock-shop-2/manifests/12-orders-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'orders'.

- It is annotated to enable Prometheus scraping with 'prometheus.io/scrape: true'.
- The Service is labeled with 'name: orders'.
- It is deployed in the 'sock-shop' namespace.
- The Service exposes port 80 and directs traffic to the same port on the target pods.
- It uses a selector to match pods with the label 'name: orders'.

`sock-shop-2/manifests/13-orders-db-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'orders-db' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'orders-db' pod running.
- The pods are selected based on the label 'name: orders-db'.
- Each pod runs a single container using the 'mongo' image.
- The container exposes port 27017, which is the default port for MongoDB.
- Security context is set to drop all capabilities and add only CHOWN, SETGID, and SETUID, with a read-only root filesystem.
- A volume named 'tmp-volume' is mounted at '/tmp' and uses an in-memory empty directory.
- The pods are scheduled to run on nodes with the operating system labeled as 'linux'.

`sock-shop-2/manifests/14-orders-db-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'orders-db'.
- It is located in the 'sock-shop' namespace.
- The Service is configured to expose port 27017.
- It targets the same port (27017) on the pods.
- The Service selects pods with the label 'name: orders-db'.

`sock-shop-2/manifests/15-payment-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'payment' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'payment' application running.
- The Deployment uses the Docker image 'weaveworksdemos/payment:0.4.3'.
- Resource limits are set for the container, with a maximum of 200m CPU and 200Mi memory, and requests for 99m CPU and 100Mi memory.
- The container listens on port 80.
- Security settings ensure the container runs as a non-root user with user ID 10001, and the root filesystem is read-only.

- The container has a liveness probe and a readiness probe, both checking the '/health' endpoint on port 80, with initial delays of 300 and 180 seconds respectively.
- The Deployment is configured to run on nodes with the Linux operating system.

`sock-shop-2/manifests/16-payment-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'payment'.
- It is annotated for Prometheus scraping, which means it is set up for monitoring.
- The Service is labeled with 'name: payment'.
- It is deployed in the 'sock-shop' namespace.
- The Service exposes port 80 and directs traffic to the same port on the selected pods.
- The Service selects pods with the label 'name: payment' to route traffic to them.

`sock-shop-2/manifests/17-queue-master-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'queue-master' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'queue-master' application running.
- The Deployment uses a container image 'weaveworksdemos/queue-master:0.3.1'.
- Environment variables are set for the container, including Java options for memory management and garbage collection.
- Resource limits and requests are defined, with a CPU limit of 300m and memory limit of 500Mi, and requests for 100m CPU and 300Mi memory.
- The container exposes port 80.
- The Deployment is configured to run on nodes with the Linux operating system.

`sock-shop-2/manifests/18-queue-master-svc.yaml`

- This manifest defines a Kubernetes Service.
- The service is named 'queue-master'.
- It is annotated for Prometheus scraping, which means it is set up to be monitored by Prometheus.
- The service is labeled with 'name: queue-master' for identification and selection purposes.
- It is deployed in the 'sock-shop' namespace.
- The service exposes port 80 and directs traffic to the same port on the target pods.
- The service uses a selector to target pods with the label 'name: queue-master'.

`sock-shop-2/manifests/19-rabbitmq-dep.yaml`

- This manifest defines a Deployment for RabbitMQ in a Kubernetes cluster.

- The Deployment is named 'rabbitmq' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the RabbitMQ pod running.
- The Deployment uses a selector to match pods with the label 'name: rabbitmq'.
- The pod template includes two containers: one for RabbitMQ and another for RabbitMQ Exporter.
- The RabbitMQ container uses the image 'rabbitmq:3.6.8-management' and exposes two ports: 15672 for management and 5672 for RabbitMQ.
- The RabbitMQ container has a security context that drops all capabilities and adds specific ones like CHOWN, SETGID, SETUID, and DAC\_OVERRIDE, and it uses a read-only root filesystem.
- The RabbitMQ Exporter container uses the image 'kbudde/rabbitmq-exporter' and exposes port 9090.
- The Deployment is configured to run on nodes with the label 'beta.kubernetes.io/os: linux'.
- Annotations are set to prevent Prometheus from scraping metrics from this pod.

`sock-shop-2/manifests/20-rabbitmq-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'rabbitmq'.
- It is annotated for Prometheus scraping on port 9090.
- The Service is labeled with 'name: rabbitmq'.
- It is deployed in the 'sock-shop' namespace.
- The Service exposes two ports: 5672 for RabbitMQ and 9090 for an exporter.
- The Service uses TCP protocol for communication.
- It selects pods with the label 'name: rabbitmq' to route traffic to.

`sock-shop-2/manifests/21-session-db-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'session-db' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'session-db' pod running.
- The pods are selected based on the label 'name: session-db'.
- Each pod runs a single container using the 'redis' image.
- The container exposes port 6379, which is commonly used by Redis.
- Security settings are applied to drop all capabilities and only add CHOWN, SETGID, and SETUID.
- The root filesystem of the container is set to be read-only for security purposes.
- The pods are scheduled to run on nodes with the operating system labeled as Linux.

#### `sock-shop-2/manifests/22-session-db-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'session-db'.
- It is located in the 'sock-shop' namespace.
- The Service listens on port 6379 and forwards traffic to the same port on the target pods.
- It uses a selector to target pods with the label 'name: session-db'.

#### `sock-shop-2/manifests/23-shipping-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'shipping' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'shipping' application running.
- The Deployment uses the Docker image 'weaveworksdemos/shipping:0.4.8'.
- Environment variables are set for the application, including 'ZIPKIN' and 'JAVA\_OPTS'.
- Resource limits and requests are defined, with a CPU limit of 300m and memory limit of 500Mi, and requests for 100m CPU and 300Mi memory.
- The application listens on port 80.
- Security context is configured to run the container as a non-root user with user ID 10001, and with a read-only root filesystem.
- The container has a volume mounted at '/tmp', which is an in-memory empty directory.
- The Deployment is scheduled to run on nodes with the Linux operating system.

#### `sock-shop-2/manifests/24-shipping-svc.yaml`

- This is a Kubernetes Service manifest.
- The service is named 'shipping'.
- It is annotated for Prometheus scraping with 'prometheus.io/scrape: true'.
- The service is labeled with 'name: shipping'.
- It is deployed in the 'sock-shop' namespace.
- The service exposes port 80 and directs traffic to the same port on the selected pods.
- It selects pods with the label 'name: shipping'.

#### `sock-shop-2/manifests/25-user-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'user' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'user' application running.
- The Deployment uses the Docker image 'weaveworksdemos/user:0.4.7'.
- Resource limits are set for the container: 300m CPU and 200Mi memory, with requests for 100m CPU and 100Mi memory.

- The container listens on port 80.
- An environment variable 'mongo' is set with the value 'user-db:27017'.
- Security context is configured to run the container as a non-root user with user ID 10001.
- The container has a read-only root filesystem and drops all capabilities except 'NET\_BIND\_SERVICE'.
- Liveness and readiness probes are configured to check the '/health' endpoint on port 80.
- The liveness probe starts after 300 seconds and checks every 3 seconds.
- The readiness probe starts after 180 seconds and checks every 3 seconds.
- The Deployment is scheduled to run on nodes with the Linux operating system.

`sock-shop-2/manifests/26-user-svc.yaml`

- This manifest defines a Kubernetes Service.
- The service is named 'user'.
- It is annotated for Prometheus scraping, which means it is set up for monitoring.
- The service is labeled with 'name: user'.
- It is deployed in the 'sock-shop' namespace.
- The service listens on port 80 and forwards traffic to the same port on the selected pods.
- The service selects pods with the label 'name: user'.

`sock-shop-2/manifests/27-user-db-dep.yaml`

- This manifest defines a Deployment in Kubernetes.
- The Deployment is named 'user-db' and is located in the 'sock-shop' namespace.
- It specifies that there should be 2 replicas of the 'user-db' pod running.
- The pods are selected based on the label 'name: user-db'.
- Each pod runs a single container using the image 'weaveworksdemos/user-db:0.3.0'.
- The container exposes port 27017, which is typically used by MongoDB.
- Security settings are applied to drop all capabilities and only add CHOWN, SETGID, and SETUID.
- The root filesystem of the container is set to be read-only for security purposes.
- A temporary volume is mounted at '/tmp' using an in-memory emptyDir volume.
- The pods are scheduled to run on nodes with the operating system labeled as 'linux'.

`sock-shop-2/manifests/28-user-db-svc.yaml`

- This manifest defines a Kubernetes Service.
- The Service is named 'user-db'.
- It is located in the 'sock-shop' namespace.
- The Service is configured to expose port 27017.

- It targets the same port (27017) on the pods it selects.
- The Service uses a selector to match pods with the label 'name: user-db'.

## Resiliency issues/weaknesses in the manifests:

### Issue #0: Missing Resource Requests

- details: Pods may not get scheduled if the cluster is under resource pressure, leading to potential downtime.
- manifests having the issues: ['sock-shop-2/manifests/03-carts-db-dep.yaml', 'sock-shop-2/manifests/07-catalogue-db-dep.yaml', 'sock-shop-2/manifests/13-orders-db-dep.yaml', 'sock-shop-2/manifests/19-rabbitmq-dep.yaml', 'sock-shop-2/manifests/21-session-db-dep.yaml', 'sock-shop-2/manifests/27-user-db-dep.yaml']
- problematic config: The deployments for carts-db, catalogue-db, orders-db, rabbitmq, session-db, and user-db do not specify resource requests.

### Issue #1: Single Replica Deployment

- details: The front-end deployment has only one replica, which can lead to downtime if the pod fails.
- manifests having the issues: ['sock-shop-2/manifests/09-front-end-dep.yaml']
- problematic config: spec.replicas: 1

### Issue #2: High Initial Delay for Liveness Probe

- details: A high initial delay for the liveness probe can delay the detection of a failed pod, leading to longer downtime.
- manifests having the issues: ['sock-shop-2/manifests/05-catalogue-dep.yaml', 'sock-shop-2/manifests/09-front-end-dep.yaml', 'sock-shop-2/manifests/15-payment-dep.yaml', 'sock-shop-2/manifests/25-user-dep.yaml']
- problematic config: livenessProbe.initialDelaySeconds: 300

### Issue #3: Missing Port Specification in Service

- details: The service may not function correctly if the port is not specified, leading to connectivity issues.
- manifests having the issues: ['sock-shop-2/manifests/02-carts-svc.yaml', 'sock-shop-2/manifests/04-carts-db-svc.yaml', 'sock-shop-2/manifests/06-catalogue-svc.yaml', 'sock-shop-2/manifests/08-catalogue-db-svc.yaml', 'sock-shop-2/manifests/12-orders-svc.yaml', 'sock-shop-2/manifests/14-orders-db-svc.yaml', 'sock-shop-2/manifests/16-payment-svc.yaml', 'sock-shop-2/manifests/18-queue-master-svc.yaml', 'sock-shop-2/manifests/22-session-db-svc.yaml', 'sock-shop-2/manifests/24-shipping-svc.yaml', 'sock-shop-2/manifests/26-user-svc.yaml', 'sock-shop-2/manifests/28-user-db-svc.yaml']
- problematic config: The port field is commented out or not specified in the service manifests.

## Application of the manifests:

### Thoughts:

The manifests provided are part of a Kubernetes deployment for a microservices-based application called 'sock-shop'. This is evident from the namespace 'sock-shop' and the naming conventions used in the manifests, such as 'carts', 'catalogue', 'orders', 'payment', 'shipping', 'user', and their respective databases. These components are typical of an e-commerce application, where users can browse a catalogue, add items to a cart, place orders, and make payments. The use of services like RabbitMQ and Redis further supports the assumption of a complex, distributed application architecture, which is common in e-commerce platforms.

### Assumed application:

The 'sock-shop' application is an e-commerce platform that allows users to browse a catalogue of items, add items to a shopping cart, place orders, and make payments. It is a microservices-based application deployed on Kubernetes, with each service handling a specific aspect of the e-commerce process, such as managing the catalogue, processing orders, handling payments, and managing user sessions.

## Summary of your instructions for Chaos Engineering:

- The Chaos-Engineering experiment must be completed within 1 minute.
- When using k6 in steady-state definition, always select a request URL from the following options (other requests are invalid):
  1. <http://front-end.sock-shop.svc.cluster.local/>
  2. <http://front-end.sock-shop.svc.cluster.local/catalogue?size=10>
  3. <http://front-end.sock-shop.svc.cluster.local/detail.html?id=<ID>> Replace <ID> with an available ID: [ 03fef6ac-1896-4ce8-bd69-b798f85c6e0b , 3395a43e-2d88-40de-b95f-e00e1502085b , 510a0d7e-8e83-4193-b483-e27e09ddc34d , 808a2de1-1aaa-4c25-a9b9-6612e8f29a38 , 819e1fbf-8b7e-4f6d-811f-693534916a8b , 837ab141-399e-4c1f-9abc-bace40296bac , a0a4f044-b040-410d-8ead-4de0446aec7e , d3588630-ad8e-49df-bbd7-3167f7efb246 , zzz4f044-b040-410d-8ead-4de0446aec7e ]
  4. <http://front-end.sock-shop.svc.cluster.local/category/>
  5. <http://front-end.sock-shop.svc.cluster.local/category?tags=<TAG>> Replace <TAG> with an available tag: [ magic , action , blue , brown , black , sport , formal , red , green , skin , geek ]
  6. <http://front-end.sock-shop.svc.cluster.local/basket.html>

## Phase 1: Hypothesis

---

## ✓ Steady state #1: carts-db-pod-count

### 🗨 Description

The first issue to address is the 'Missing Resource Requests' in the 'carts-db' Deployment. This is a critical issue because without resource requests, the pods may not get scheduled if the cluster is under resource pressure, leading to potential downtime. To verify the steady state, we can measure the number of 'carts-db' pods running, ensuring it matches the expected replica count of 2. This will help us understand if the pods are being scheduled correctly despite the missing resource requests.

### 🔍 Current state inspection

To inspect the 'carts-db-pod-count', I will use the Kubernetes API to check the number of running pods for the 'carts-db' deployment. This is because the task involves verifying the number of pods, which is a Kubernetes resource state, and the K8s API is suitable for this purpose. The deployment is in the 'sock-shop' namespace, and the expected replica count is 2. I will implement a Python script using the K8s client library to check the pod count every second for a duration of 5 seconds.

tool: `k8s` duration: `5s`

Inspection script (Python) `k8s_carts-db-pod-count.py` :

```
import os
import time
from kubernetes import client, config

# Load Kubernetes configuration based on the environment
if os.getenv('KUBERNETES_SERVICE_HOST'):
    config.load_incluster_config()
else:
    config.load_kube_config()

v1 = client.AppsV1Api()

def check_carts_db_pod_count(namespace, expected_count):
    pod_count = 0
    try:
        resp = v1.read_namespaced_deployment(name='carts-db', namespace=namespace)
        pod_count = resp.status.replicas
    except client.exceptions.ApiException as e:
```

```

        print(f"Exception when calling AppsV1Api->read_namespaced_de
return pod_count

def main(duration):
    namespace = 'sock-shop'
    expected_count = 2
    for _ in range(duration):
        pod_count = check_carts_db_pod_count(namespace, expected_count)
        print(f"Current 'carts-db' pod count: {pod_count}")
        if pod_count == expected_count:
            print("Pod count matches expected count.")
        else:
            print("Pod count does not match expected count.")
        time.sleep(1)

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser(description='Check carts-db pod count')
    parser.add_argument('--duration', type=int, default=5, help='Duration of the check')
    args = parser.parse_args()
    main(args.duration)

```

Pod `k8scarts-db-pod-count-pod` has completed successfully.

The inspection script's results (current states) are as follows:

```

Current 'carts-db' pod count: 2
Pod count matches expected count.
Current 'carts-db' pod count: 2
Pod count matches expected count.
Current 'carts-db' pod count: 2
Pod count matches expected count.
Current 'carts-db' pod count: 2
Pod count matches expected count.
Current 'carts-db' pod count: 2
Pod count matches expected count.
Current 'carts-db' pod count: 2
Pod count matches expected count.

```

## Threshold

The steady state we are examining is the number of 'carts-db' pods running, which should match the expected replica count of 2. The current state shows that the pod count consistently matches the expected count of 2 over multiple checks. Given that the system is expected to maintain 2 replicas, the threshold should ensure that at least 2 pods are running to maintain redundancy and availability. To account for potential

fluctuations, such as during pod restarts or updates, a reasonable tolerance can be applied. For example, allowing a brief period where the pod count might drop to 1, but it should recover quickly. Therefore, the threshold can be defined as maintaining at least 2 'carts-db' pods running 95% of the time within a 1-minute monitoring period. This allows for brief fluctuations while ensuring the system remains stable and available.

At least 2 'carts-db' pods must be running 95% of the time within a 1-minute monitoring period.

### Unit test to validate the steady state

To add the threshold assertion to the inspection Python script, I will implement a loop that checks the number of 'carts-db' pods running every second for the specified duration. I will count the number of times the pod count matches the expected count of 2. At the end of the duration, I will calculate the percentage of time the pod count was as expected. I will then assert that this percentage is at least 95%, which is the threshold defined for the steady state. This approach allows for brief fluctuations while ensuring the system remains stable and available.

```
import os
import time
import argparse
from kubernetes import client, config
from unittest_base import K8sAPIBase

class TestCartsDBPodCount(K8sAPIBase):
    def __init__(self):
        super().__init__()
        self.v1_apps = client.AppsV1Api()

    def check_carts_db_pod_count(self, namespace, expected_count):
        pod_count = 0
        try:
            resp = self.v1_apps.read_namespaced_deployment(name='carts-db', namespace=namespace)
            pod_count = resp.status.replicas
        except client.exceptions.ApiException as e:
            print(f"Exception when calling AppsV1Api->read_namespaced_deployment: {e}")
        return pod_count

    def test_steady_state(self, duration):
        namespace = 'sock-shop'
        expected_count = 2
        successful_checks = 0
```

```

for _ in range(duration):
    pod_count = self.check_carts_db_pod_count(namespace, expected_count)
    print(f"Current 'carts-db' pod count: {pod_count}")
    if pod_count == expected_count:
        successful_checks += 1
    time.sleep(1)

# Calculate the percentage of successful checks
success_percentage = (successful_checks / duration) * 100
print(f"Success percentage: {success_percentage}%")

# Assert that the success percentage is at least 95%
assert success_percentage >= 95, f"Pod count was not stable"

def main():
    parser = argparse.ArgumentParser(description='Test carts-db pod count')
    parser.add_argument('--duration', type=int, default=60, help='Duration in seconds')
    args = parser.parse_args()

    test = TestCartsDBPodCount()
    test.test_steady_state(args.duration)

if __name__ == '__main__':
    main()

```

Pod `unittestcarts-db-pod-countmod0-pod` has completed successfully.

The inspection script's results (current states) are as follows:

```

Current 'carts-db' pod count: 2
Current 'carts-db' pod count: 2
Current 'carts-db' pod count: 2
Current 'carts-db' pod count: 2
Current 'carts-db' pod count: 2
Success percentage: 100.0%

```

## Steady state #2: front-end-pod-count



 Description

The next issue to address is the 'Single Replica Deployment' in the 'front-end' Deployment. This is a critical issue because having only one replica can lead to downtime if the pod fails. To verify the steady state, we can measure the number of 'front-end' pods running, ensuring it matches the expected replica count of 1. This will help us understand if the pod is being scheduled correctly and remains available despite being a single replica. Monitoring the pod count will allow us to verify the system's resilience to potential failures.

## Current state inspection

To inspect the 'front-end-pod-count', I will use the K8s API. This is because the task involves checking the number of pods running for the 'front-end' deployment, which is a Kubernetes resource. The K8s API is suitable for querying the current state of Kubernetes resources, such as deployments and pods. I will write a Python script using the Kubernetes client library to check the number of 'front-end' pods running in the 'sock-shop' namespace. The script will run for a duration of 5 seconds, checking the pod count every second, and will print a summary of the results at the end.

tool: `k8s` duration: `5s`

Inspection script (Python) `k8s_front-end-pod-count.py` :

```
import os
import time
import argparse
from kubernetes import client, config

def check_front_end_pod_count(namespace, duration):
    # Load Kubernetes configuration based on the environment
    if os.getenv('KUBERNETES_SERVICE_HOST'):
        config.load_incluster_config()
    else:
        config.load_kube_config()

    v1 = client.CoreV1Api()
    pod_label_selector = 'name=front-end'

    for _ in range(duration):
        pods = v1.list_namespaced_pod(namespace=namespace, label_selector=pod_label_selector)
        pod_count = len(pods.items)
        print(f'Current front-end pod count: {pod_count}')
        time.sleep(1)

    print('Finished checking front-end pod count.')

if __name__ == '__main__':
```

```
parser = argparse.ArgumentParser(description='Check front-end pod count')
parser.add_argument('--duration', type=int, default=5, help='Duration in seconds')
args = parser.parse_args()
```

```
check_front_end_pod_count(namespace='sock-shop', duration=args.duration)
```

Pod `k8sfront-end-pod-count-pod` has completed successfully.

The inspection script's results (current states) are as follows:

```
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Finished checking front-end pod count.
```

## 🚩 Threshold

The current state of the system shows that the front-end pod count is consistently at 1, which matches the expected replica count as per the deployment configuration. Given that the system is currently stable with a single replica, the threshold should ensure that at least 1 front-end pod is running at all times. However, to account for potential fluctuations or brief disruptions, a reasonable tolerance should be included. Since the experiment must be completed within 1 minute, we can set a threshold that allows for a brief period where the pod count might temporarily drop below 1, but it should recover quickly. Therefore, the threshold can be defined as having at least 1 front-end pod running 95% of the time during the 1-minute monitoring period. This allows for a brief 3-second window where the pod count might be 0, but it should not persist beyond that.

At least 1 front-end pod must be running 95% of the time during a 1-minute monitoring period.

## 📄 Unit test to validate the steady state

To verify the steady state of the front-end pod count, I will implement a unit test that checks if at least 1 front-end pod is running 95% of the time during the specified duration. This will be done by counting the number of times the pod count is 1 or more and ensuring it meets the 95% threshold. The test will use a for-loop to check the pod count every second for the specified duration, and at the end, it will assert whether the

threshold is met. The threshold is represented as a percentage (95%) and will be calculated based on the duration provided by the user.

```
import os
import time
import argparse
from kubernetes import client, config
from unittest_base import K8sAPIBase

class TestFrontEndPodCount(K8sAPIBase):
    def __init__(self, namespace, duration):
        super().__init__()
        self.namespace = namespace
        self.duration = duration

    def test_front_end_pod_count(self):
        pod_label_selector = 'name=front-end'
        successful_checks = 0

        # Check the pod count every second for the specified duration
        for _ in range(self.duration):
            pods = self.v1.list_namespaced_pod(namespace=self.namespace, label_selector=pod_label_selector)
            pod_count = len(pods.items)
            print(f'Current front-end pod count: {pod_count}')

            # Increment successful checks if pod count is 1 or more
            if pod_count >= 1:
                successful_checks += 1

            time.sleep(1)

        # Calculate the percentage of successful checks
        success_rate = (successful_checks / self.duration) * 100
        print(f'Success rate: {success_rate}%')

        # Assert that the success rate meets the 95% threshold
        assert success_rate >= 95, f"Front-end pod count did not meet threshold"

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Test front-end pod count')
    parser.add_argument('--duration', type=int, default=60, help='Duration in seconds')
    args = parser.parse_args()

    test = TestFrontEndPodCount(namespace='sock-shop', duration=args.duration)
    test.test_front_end_pod_count()
```

Pod `unittestfront-end-pod-countmod0-pod` has completed successfully.

The inspection script's results (current states) are as follows:

```
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Current front-end pod count: 1
Success rate: 100.0%
```

## Fault definition

### ✓ Scenario: Black Friday Sale

#### 🗨 Description

During a Black Friday sale, the system is expected to handle a significant increase in traffic and load. This event will test the system's ability to maintain steady states under high demand. The system's weaknesses, such as missing resource requests and single replica deployments, can lead to potential downtime or degraded performance. To simulate this event, we will inject faults that target these weaknesses. First, we will use `StressChaos` to simulate high CPU and memory usage on the 'carts-db' pods, which lack resource requests, to see if they can maintain the required pod count. Next, we will use `PodChaos` to kill the single replica 'front-end' pod to test its resilience and recovery. Finally, we will introduce `NetworkChaos` to simulate network latency on the 'front-end' service, which will test the system's ability to handle network issues during high traffic. This sequence of fault injections will simulate the phenomena of a Black Friday sale, where high load, potential failures, and network issues are common.

#### 🎯 Fault-injection sequence

```
StressChaos ({'namespace': 'sock-shop', 'label': 'name=carts-db'}) → PodChaos
({'namespace': 'sock-shop', 'label': 'name=front-end'}) → NetworkChaos ({'namespace':
'sock-shop', 'label': 'name=front-end'})
```

#### ⚙ Detailed fault parameters

Detailed parameters of `StressChaos` ({'namespace': 'sock-shop', 'label': 'name=carts-db'})

```

{
  "mode" : "all"
  "selector" : {
    "namespaces" : [
      0 : "sock-shop"
    ]
    "labelSelectors" : {
      "name" : "carts-db"
    }
  }
  "stressors" : {
    "cpu" : {
      "workers" : 2
      "load" : 100
    }
    "memory" : {
      "workers" : 2
      "size" : "256MB"
    }
  }
  "containerNames" : [
    0 : "carts-db"
  ]
}

```

Detailed parameters of **PodChaos** ({'namespace': 'sock-shop', 'label': 'name=front-end'})

```

{
  "action" : "pod-kill"
  "mode" : "one"
  "selector" : {
    "namespaces" : [
      0 : "sock-shop"
    ]
    "labelSelectors" : {
      "name" : "front-end"
    }
  }
}

```

Detailed parameters of **NetworkChaos** ({'namespace': 'sock-shop', 'label': 'name=front-end'})

```

{
  "action" : "delay"
  "mode" : "all"
  "selector" : {
    "namespaces" : [
      0 : "sock-shop"
    ]
    "labelSelectors" : {
      "name" : "front-end"
    }
  }
  "direction" : "to"
  "device" : "eth0"
  "delay" : {
    "latency" : "100ms"
    "jitter" : "10ms"
    "correlation" : "50"
  }
}

```

## Phase 2: Chaos Experiment

CE experiment Planning Completed!

### Chaos Engineering Experiment Plan

#### Time Schedule

To design a Chaos Engineering experiment that fits within the 1-minute constraint, we need to allocate time efficiently across the three phases: pre-validation, fault injection, and post-validation. Given the steady states and the fault scenario, we should ensure that each phase is long enough to gather meaningful data while staying within the total time limit.

1. **Pre-validation Phase:** This phase is crucial to ensure that the system is in a steady state before we introduce any faults. We need to verify that the 'carts-db' and 'front-end' pods are running as expected. Given the 1-minute total time constraint, we can allocate 15 seconds for this phase. This should be sufficient to run the steady

state checks for both 'carts-db' and 'front-end' pods, as the checks are designed to run quickly.

2. **Fault Injection Phase:** This is the core of the experiment where we introduce the faults. We have three faults to inject: StressChaos on 'carts-db', PodChaos on 'front-end', and NetworkChaos on 'front-end'. We need to observe the system's behavior under these conditions. Allocating 30 seconds for this phase allows us to introduce each fault and observe the immediate impact on the system. This duration is a compromise between observing the effects and staying within the time limit.
3. **Post-validation Phase:** After the faults are removed, we need to ensure that the system returns to its steady state. This phase should verify that the 'carts-db' and 'front-end' pods are back to their expected counts. Allocating 15 seconds for this phase allows us to run the steady state checks again to confirm recovery.

In summary, the time allocation is 15 seconds for pre-validation, 30 seconds for fault injection, and 15 seconds for post-validation, totaling 60 seconds.

Total experiment time: 60s

Pre-validation Phase: 15s

Fault-injection Phase: 30s

Post-validation Phase: 15s

### Pre-validation Phase (15s)

In the pre-validation phase, we need to ensure that the system is in a steady state before we proceed with fault injection. This involves verifying that the key components of the system are operating as expected. Given the constraints of a 15-second total time for this phase, we need to efficiently schedule the unit tests to verify the steady states.

The two steady states we need to verify are:

1. `carts-db-pod-count` : Ensure that at least 2 'carts-db' pods are running 95% of the time within a 1-minute monitoring period.
2. `front-end-pod-count` : Ensure that at least 1 'front-end' pod is running 95% of the time during a 1-minute monitoring period.

Since we only have 15 seconds for the pre-validation phase, we will perform a quick check to ensure that the current pod counts meet the expected values. This will not be a full 1-minute check but a snapshot to ensure that the system is in a steady state at the start of the experiment.

The approach is to run both unit tests simultaneously to maximize the use of the limited time. Each test will have a grace period of 0 seconds and a duration of 15 seconds, allowing us to quickly verify the current state of the system. This approach ensures that we can confirm the system's readiness for the chaos experiment within the given time constraint.

- Verified Steady State #0: `cars-db-pod-count`
  - Workflow Name: `pre-unittest-cars-db-pod-count`
  - Grace Period: `0s`
  - Duration: `15s`
- Verified Steady State #1: `front-end-pod-count`
  - Workflow Name: `pre-unittest-front-end-pod-count`
  - Grace Period: `0s`
  - Duration: `15s`

### Fault-injection Phase (30s)

In this fault-injection phase, we aim to simulate a Black Friday sale scenario by introducing stress and disruptions to the system. The phase is designed to last 30 seconds, and we will inject three types of faults sequentially to observe the system's behavior under stress. The sequence and timing of the fault injections are crucial to effectively simulate the scenario and observe the system's resilience.

1. **StressChaos on 'cars-db' Pods:** We will start by injecting a StressChaos fault targeting the 'cars-db' pods. This fault will simulate high CPU and memory usage, which is typical during a high-traffic event like Black Friday. The stress will be applied for 10 seconds, starting immediately at the beginning of the phase (grace period of 0s). This will help us observe if the 'cars-db' pods can maintain the required pod count despite the missing resource requests.
2. **PodChaos on 'front-end' Pod:** After the StressChaos, we will introduce a PodChaos fault to kill the single replica 'front-end' pod. This fault will be injected at the 10-second mark (grace period of 10s) and will last for 10 seconds. This will test the system's ability to recover from a pod failure, especially since the 'front-end' deployment has only one replica.
3. **NetworkChaos on 'front-end' Service:** Finally, we will inject a NetworkChaos fault to introduce network latency on the 'front-end' service. This fault will be injected at the 20-second mark (grace period of 20s) and will last for 10 seconds. This simulates network issues that might occur during high traffic, testing the system's ability to handle such disruptions.

The faults are injected sequentially to ensure that each fault's impact can be observed independently, while also simulating a realistic scenario where multiple issues might occur in quick succession during a high-demand event. The timing is staggered to allow for clear observation of each fault's effects and the system's response.

- Injected Faults #0: `StressChaos`
  - Workflow Name: `fault-stresschaos`
  - Grace Period: `0s`
  - Duration: `10s`
- Injected Faults #1: `PodChaos`
  - Workflow Name: `fault-podchaos`
  - Grace Period: `10s`
  - Duration: `10s`
- Injected Faults #2: `NetworkChaos`
  - Workflow Name: `fault-networkchaos`
  - Grace Period: `20s`
  - Duration: `10s`

### Post-validation Phase (15s)

In the post-validation phase, we need to ensure that the system has returned to its steady states after the fault injection. Given the constraints of a 15-second total time for this phase, we will focus on quickly verifying the two defined steady states: 'carts-db-pod-count' and 'front-end-pod-count'. The approach is to execute these unit tests simultaneously to maximize the use of the limited time available. Each unit test will have a short grace period to allow the system to stabilize after the fault injection, followed by a brief duration to verify the steady state. The thought process is to ensure that both critical components, 'carts-db' and 'front-end', are functioning as expected, which will indicate that the system has recovered from the injected faults.

- Verified Steady State #0: `carts-db-pod-count`
  - Workflow Name: `post-unittest-carts-db-pod-count`
  - Grace Period: `2s`
  - Duration: `6s`
- Verified Steady State #1: `front-end-pod-count`
  - Workflow Name: `post-unittest-front-end-pod-count`
  - Grace Period: `2s`
  - Duration: `6s`

### Summary

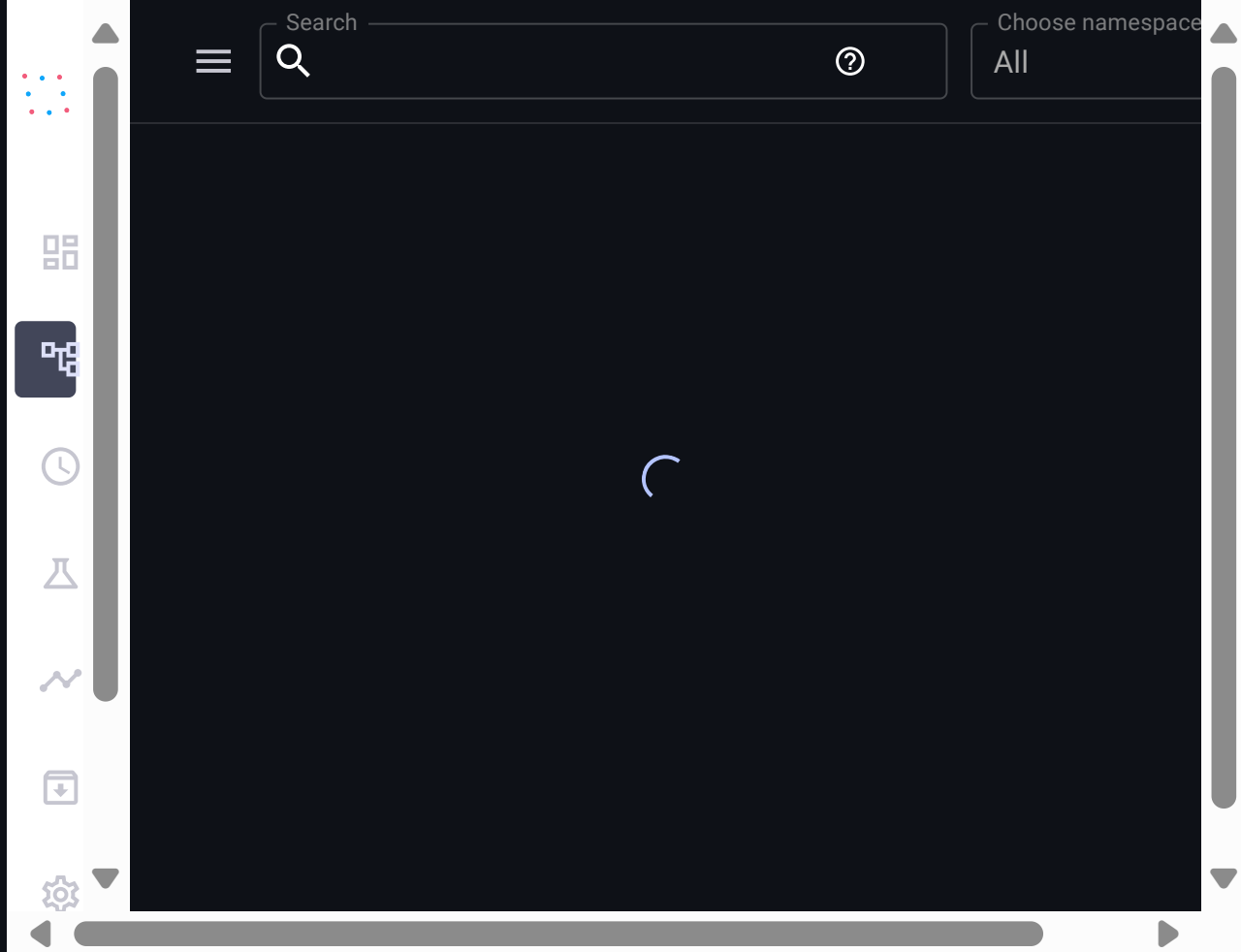
The Chaos Engineering experiment is structured into three phases: pre-validation, fault injection, and post-validation, all within a total duration of 60 seconds.

In the pre-validation phase, which lasts for 15 seconds, the system's steady state is verified by checking the 'carts-db' and 'front-end' pods. Both unit tests, 'carts-db-pod-count' and 'front-end-pod-count', are executed simultaneously with no delay at the start of this phase. Each test runs for the entire 15 seconds to ensure the system is ready for the chaos experiment.

The fault injection phase spans 30 seconds and involves three sequential fault injections. First, a StressChaos fault is applied to the 'carts-db' pods immediately at the start of the phase, lasting for 10 seconds. Next, a PodChaos fault is introduced to the 'front-end' pod at the 10-second mark, also lasting for 10 seconds. Finally, a NetworkChaos fault is injected on the 'front-end' service at the 20-second mark, continuing for 10 seconds. This staggered approach allows for independent observation of each fault's impact.

In the post-validation phase, which also lasts 15 seconds, the system's recovery is verified. The unit tests 'carts-db-pod-count' and 'front-end-pod-count' are executed simultaneously, each starting after a 2-second delay to allow for stabilization. Both tests run for 6 seconds to confirm that the system has returned to its steady state after the faults are removed.

**Completed the chaos experiment!**



Your k8s yaml already has good resilience!!!

## Phase EX: Postprocessing

### Summary of your k8s yaml

The Chaos Engineering cycle for the 'sock-shop' application involves a detailed process to test the system's resilience under simulated fault conditions. The application is a microservices-based e-commerce platform deployed on Kubernetes, consisting of various components such as 'carts', 'catalogue', 'orders', 'payment', 'shipping', 'user', and their respective databases. The cycle begins with understanding the user inputs, which include Kubernetes manifests defining the deployment and service configurations for each component. These manifests specify details such as the number of replicas, resource limits, security settings, and service ports.

The cycle identifies several resiliency issues in the system, including missing resource requests in some deployments, a single replica deployment for the 'front-end', high initial delays for liveness probes, and missing port specifications in service manifests. These issues could lead to potential downtime or degraded performance under stress.

The Chaos Engineering experiment is designed to test the hypothesis that the system maintains its steady states even when faults are injected. Two steady states are defined: ensuring at least 2 'carts-db' pods and 1 'front-end' pod are running 95% of the time during a 1-minute monitoring period. Python scripts using the Kubernetes API are provided to verify these steady states.

The fault scenario simulates a Black Friday sale, introducing high load and potential failures. Chaos Mesh is used to inject faults, including StressChaos on 'carts-db' to simulate high CPU and memory usage, PodChaos to kill the 'front-end' pod, and NetworkChaos to introduce network latency on the 'front-end' service.

The experiment is divided into three phases: pre-validation, fault injection, and post-validation, each with specific time allocations to fit within a 1-minute constraint. Pre-validation ensures the system is in a steady state, fault injection introduces the faults, and post-validation checks if the system returns to its steady state.

The experiment is executed using a Chaos Mesh Workflow file, which automates the process according to the plan. The first try of the experiment results in all unit tests passing, indicating that the system maintained its steady states throughout the experiment. This suggests that the system is resilient to the simulated faults, although further improvements could be made to address the identified issues.

[Download output \(.zip\)](#)