Preguntas Proyecto 2

¿Cómo funcionan las Golden Metrics?

Las métricas de oro (o "señales de oro") son las métricas de primera línea para saber si la aplicación está funcionando como se esperaba. Estas métricas brindan una señal de un vistazo sobre el estado de un servicio, sin necesidad de saber qué hace realmente el servicio.

La latencia que mide que tan lento o rápido es el servicio (tiempo en el que atiende las solicitudes)

El tráfico brinda una descripción general de cuánta demanda se coloca en el servicio, da una idea de lo demandado que esta un servicio (cantidad de solicitudes por segundo)

Errores que es la cantidad de solicitudes fallidas (se suele combinar con el tráfico general para generar una "tasa de exito")

saturación que es una medida de la carga del sistema, en función de sus limitaciones principales (memoria, cpu etc.)

según Linkerd: Tasa de éxito, Este es el porcentaje de solicitudes exitosas durante una ventana de tiempo (1 minuto por defecto en Linkerd).

¿Como pueden interpretarse las 7 pruebas utilizadas en base a las gráficas y métricas que muestra Linkerd y grafana?

En Linkerd se puede tomar la latencia de cada aplicación de mensajería y se puede ver que en algunos periodos de tiempo se tiene estabilidad en la latencia y otros donde se vuelve inestable, de igual manera en grafana se puede tomar la gráfica de volúmenes donde cuando las líneas bajan se interpreta que los datos no están llegando a alguna aplicación de mensajería.

¿Qué patrones de conducta fueron descubiertos?

Sin el experimento el success rate fue del 100% Con el experimento el success rate fue menor al 100% oscilando entre el rango de 45% a 60% Cuando se envían datos a alguna aplicación de mensajería el request volumen disminuye

¿Qué sistema de mensajería es más rápido? ¿Por qué?

El más rápido es PubSub por su capacidad de manejar hasta 81 mil sets por segundo y 110 mil gets por segundo.

¿Cuántos recursos utiliza cada sistema de mensajería?

Kafka

```
$ kubectl top pod grpc-deployment-kafka-666d6f8476-sdknm -n squidgame --containers
POD                                  NAME           CPU(cores)   MEMORY(bytes)
grpc-deployment-kafka-666d6f8476-sdknm   grpcclient     1m           29Mi
grpc-deployment-kafka-666d6f8476-sdknm   grpcserver     9m           53Mi
grpc-deployment-kafka-666d6f8476-sdknm   linkerd-proxy  1m           4Mi
```

```
$ kubectl top pod kafkaworker-86678d7958-5cqg4 -n squidgame --containers
POD                           NAME           CPU(cores)   MEMORY(bytes)
kafkaworker-86678d7958-5cqg4  kafkaworker    9m           8Mi
kafkaworker-86678d7958-5cqg4  linkerd-proxy  2m           6Mi
```

RabbitMQ

```
$ kubectl top pod grpc-deployment-rabbit-6b6b4bbf7d-c5zss -n squidgame --containers
POD                                  NAME           CPU(cores)   MEMORY(bytes)
grpc-deployment-rabbit-6b6b4bbf7d-c5zss  grpcclient     0m           28Mi
grpc-deployment-rabbit-6b6b4bbf7d-c5zss  grpcserver     1m           38Mi
grpc-deployment-rabbit-6b6b4bbf7d-c5zss  linkerd-proxy  1m           4Mi
```

```
$ kubectl top pod rabbitworker-77f5667654-g6g7n -n squidgame --containers
POD                            NAME           CPU(cores)   MEMORY(bytes)
rabbitworker-77f5667654-g6g7n  linkerd-proxy  2m           7Mi
rabbitworker-77f5667654-g6g7n  rabbitworker   12m          8Mi
```

Pubsub

```
$ kubectl top pod grpc-deployment-pubsub-68b5d9bdbd-x5xs2 -n squidgame --containers
POD                                  NAME           CPU(cores)   MEMORY(bytes)
grpc-deployment-pubsub-68b5d9bdbd-x5xs2  grpcclient     1m           31Mi
grpc-deployment-pubsub-68b5d9bdbd-x5xs2  grpcserver     1m           50Mi
grpc-deployment-pubsub-68b5d9bdbd-x5xs2  linkerd-proxy  1m           4Mi
```

```
$ kubectl top pod pubsubworker-fc575ddf-cfzg7 -n squidgame --containers
POD                            NAME           CPU(cores)   MEMORY(bytes)
pubsubworker-fc575ddf-cfzg7  linkerd-proxy  2m           7Mi
pubsubworker-fc575ddf-cfzg7  pubsubworker   11m          15Mi
```

¿Cuáles son las ventajas y desventajas de cada sistema?

Kafka

Ventajas

- Buena escalabilidad
- Tolerancia a fallos
- Plataforma de streaming de eventos
- Multi-tenant
- Capaz de procesar en tiempo real

- Enfocado a proyectos Big Data.

Desventajas

- Dependencia con Apache Zookeeper
- Sin enrutamiento de mensajes
- Carece de componentes de monitorización

RabbitMQ

Ventajas

- Adecuado para muchos protocolos de mensajería
- Interfaz moderna e intuitiva
- Flexibilidad y plugins disponibles
- Herramientas de desarrollo

Desventajas

- No es transaccional por defecto
- Erlang para desarrollo

Pubsub

Ventajas

- Permite la carga de imágenes y comentarios en tiempo real sobre esas imágenes.

¿Cuál es el mejor sistema de mensajería?

Cada sistema de mensajería está diseñado para distintas situaciones, por lo cual no existe una mejor que otra.

¿Cuál de las dos bases de datos se desempeña mejor y por qué?

Redis es mejor que MongoDb en las lecturas para todo tipo de cargas de trabajo, también es mejor para guardar los datos conforme la carga de trabajo vaya incrementando. Mongo usa todos los núcleos del sistema, Redis logra más ejecutándose sobre un solo núcleo sin saturarlos, aunque a su vez consuma mayor memoria RAM que Mongodb para la misma cantidad de almacenamiento

¿Cómo se refleja en los dashboards de Linkerd los experimentos de Chaos Mesh?

Se ven reflejados en la latencia y en el volumen rate además del success rate ya que ciertos experimentos dejan abajo un servicio por mucho tiempo y esto hace que el success rate disminuya y se encuentre bastante bajo.

¿En qué se diferencia cada uno de los experimentos realizados?

En que unos afectan los pod como pod failure o pod kill y otros los containers como container kill, afectan diferente parte de la infraestructura del proyecto, tomando de ejemplo pod kill y container kill, con pod kill se mata el pod en el que se están ejecutando los servicios de toda esa aplicación de mensajería, en cambio sí se elimina solo un container aún siguen activas el resto de partes de esa aplicación de mensajería que estaba en ese pod, por lo que tarda menos tiempo en restaurar el servicio, caso contrario si se realiza pod kill ya que se está quitando todo el servicio y se tendrá que levantar todos los containers de ese pod para brindar el servicio de esa aplicación de mensajería.

¿Cuál de todos los experimentos es el más dañino?

Pod kill, con pod kill se mata el pod en el que se están ejecutando los containers de toda esa aplicación de mensajería, si la aplicación de mensajería usaba 3 containers todo se ha dado de baja, por lo que tarda más tiempo en restaurar el servicio, ya que se está quitando todo el servicio y se tendrá que lenvantar todos los containers de ese pod para brindar el servicio de esa aplicación de mensajería nuevamente.

Anexos

Pod Kill

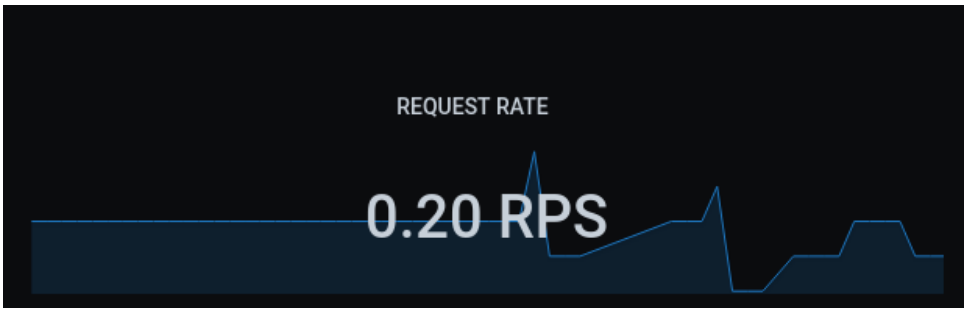
Experimento:

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  name: experiment1
  namespace: chaos-testing
#  annotations:
#    experiment.chaos-mesh.org/pause: 'false'
spec:
  schedule: '@every 20s'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: PodChaos
  podChaos:
    selector:
      namespaces:
        - squidgame
      labelSelectors:
        app: grpc-deployment-pubsub
    mode: one
    action: pod-kill
    duration: 1m
    gracePeriod: 0

```

Resultado:

```
grpc-deployment-pubsub-68b5d9bdbd-chlwv      3/3     Terminating   0          111m
grpc-deployment-pubsub-68b5d9bdbd-chlwv      3/3     Terminating   0          111m
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      0/3     Pending       0          0s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      0/3     Pending       0          0s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      0/3     Init:0/1      0          0s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      0/3     PodInitializing   0          2s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      2/3     Running           0          5s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      3/3     Running           0          14s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      3/3     Terminating       0          19s
grpc-deployment-pubsub-68b5d9bdbd-wb7rf      3/3     Terminating       0          19s
grpc-deployment-pubsub-68b5d9bdbd-xlp4h      0/3     Pending           0          0s
grpc-deployment-pubsub-68b5d9bdbd-xlp4h      0/3     Pending           0          0s
grpc-deployment-pubsub-68b5d9bdbd-xlp4h      0/3     Init:0/1          0          0s
```



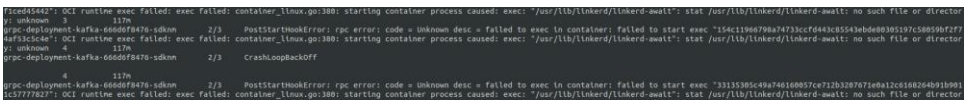
POD Failure

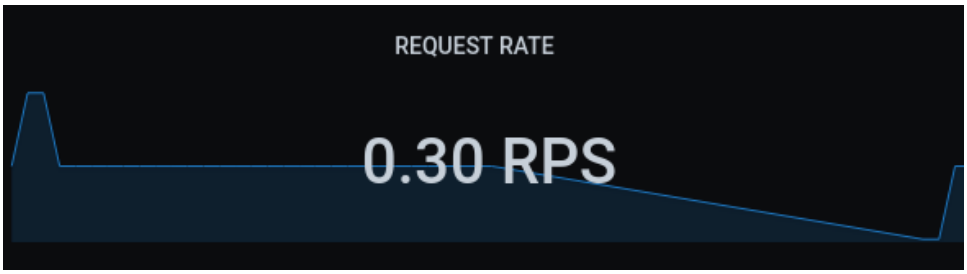
Experimento:

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  name: experiment2
  namespace: chaos-testing
#  annotations:
#    experiment.chaos-mesh.org/pause: 'false'
spec:
  schedule: '@every 10s'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: PodChaos
  podChaos:
    selector:
      namespaces:
        - squidgame
      labelSelectors:
        app: grpc-deployment-kafka
    mode: one
    action: pod-failure
    duration: 1m
    gracePeriod: 0

```

Resultado:





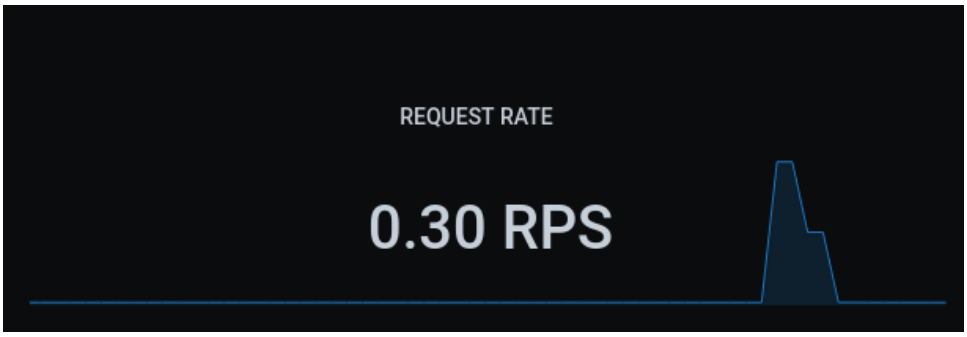
Network Emulation

Experimento:

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  name: experiment4
  namespace: chaos-testing
#  annotations:
#    experiment.chaos-mesh.org/pause: 'false'
spec:
  schedule: '@every 15s'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: NetworkChaos
  networkChaos:
    selector:
      namespaces:
        - squidgame
      labelSelectors:
        app: grpc-deployment-rabbit
    mode: one
    action: delay
    delay:
      latency: '10ms'
      correlation: '100'
      jitter: '0ms'
    duration: 1m

```

Resultado:


REQUEST RATE
0.30 RPS