

KubeCon

CloudNativeCon

North America 2024





KubeCon



CloudNativeCon

North America 2024

# Can Your Kubernetes Network Handle the Heat?

## Building Resilience with AI Chaos



KubeCon



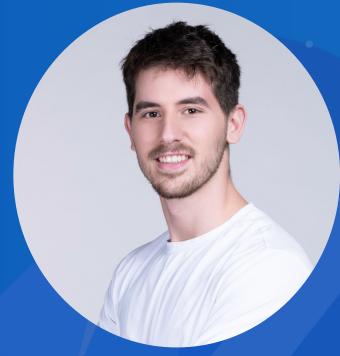
CloudNativeCon

North America 2024



Surya Seetharaman

Principal Software Engineer @ Red Hat  
SIG-Network Contributor



Lior Lieberman

Site Reliability Engineer @ Google  
Ingress2gateway Maintainer

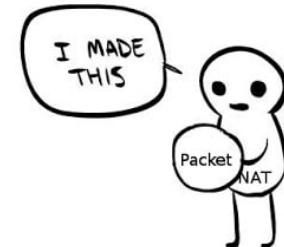
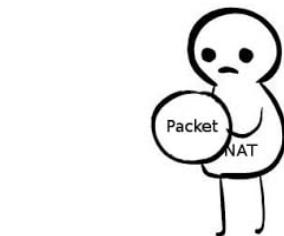
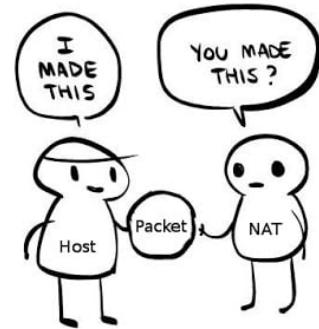


# Networking is Hard...



# Cloud Networking is Harder...

- Multiple regions, zones, and data centers
- Complex environments, VPC peering, and routing rules
- Encryption, security and tenant isolation

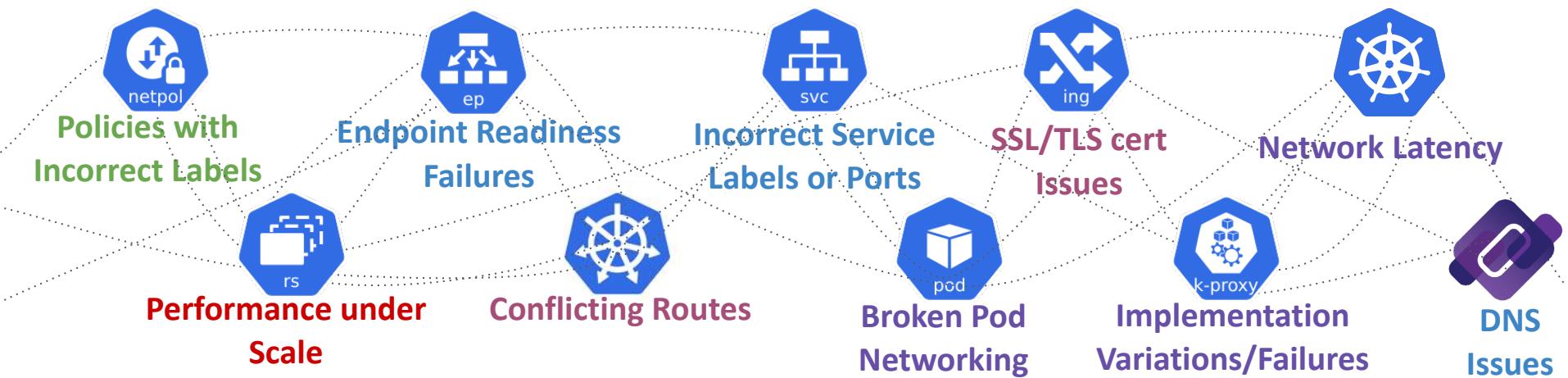


taken from: <https://dev.to/nefikras/iptables-a-beast-worth-training-a-firewall-a-nat-router-a-port-forwarder-an-lb-anti-dos-a-logger-for-free-5157>

# Kubernetes Networking is Hardest...



# What could go wrong?



- Develop abstraction for complex configurations
- Define conformance requirements
- Own common infrastructure and platform components
- Maintain security boundaries and RBACs

# Testing Network Reliability is Important

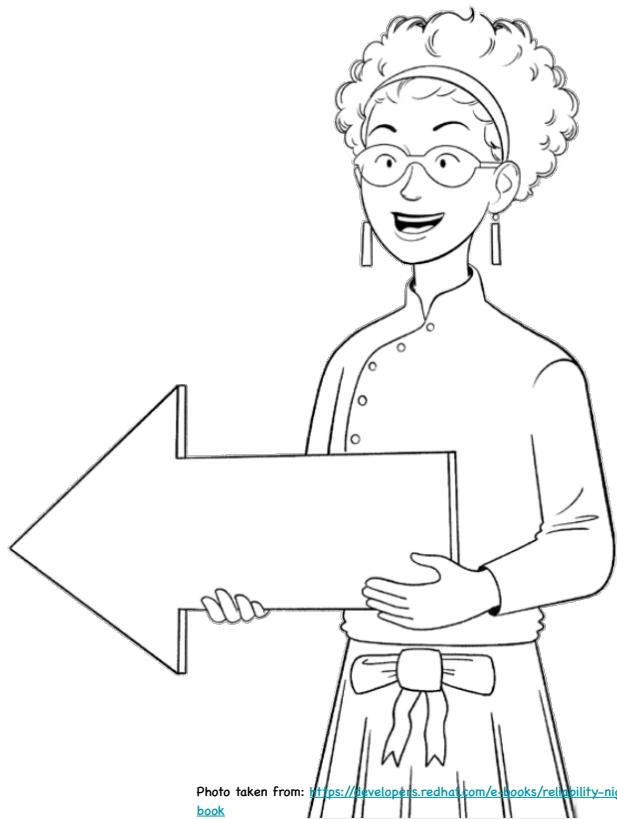
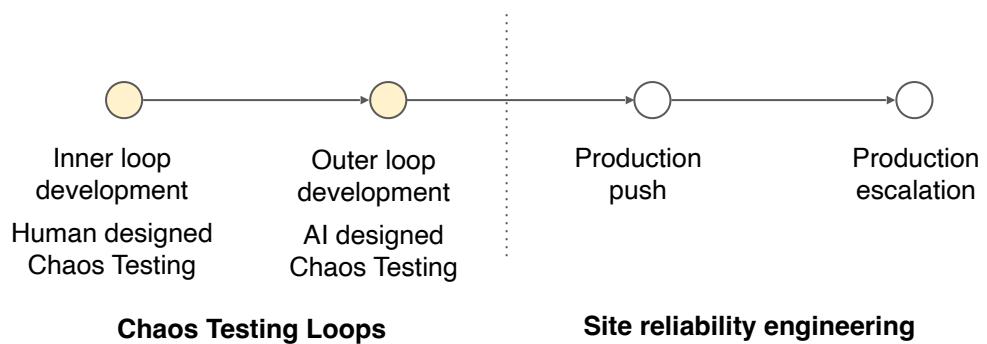
# What can we do?

- Have metrics, alerts, observability baked in...



# What can we do? Reactive Stance... (Not Enough)

- Have metrics, alerts, observability baked in...
  - It is critical but relying only on these to uncover issues is not enough!
- Shifting Left is paramount



# What can we do? Proactive Stance...++

- Use chaos testing to be prepared for the unexpected heat!!
- We have many CNCF projects to do Kubernetes Chaos and Scale Testing



**Litmus**



**krkn**



**Chaos Mesh**



**kube-burner**

# What can we do? Proactive Stance...++

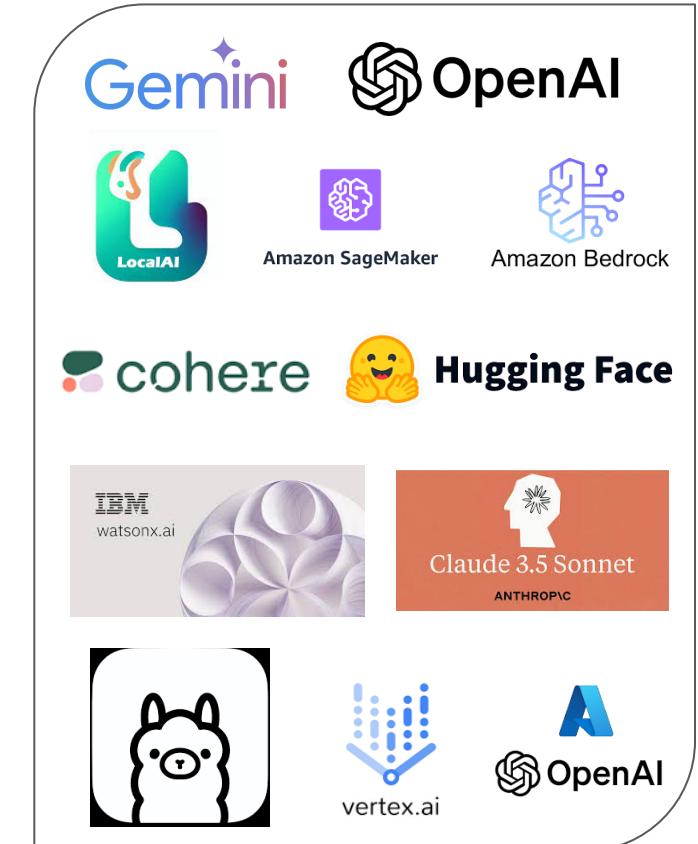
- Use chaos testing to be prepared for the unexpected heat!!
- We have many CNCF projects to do Kubernetes Chaos and Scale Testing
- However human designed chaos experiments have their limitations
  - because we are not good with being random - we design based on our knowledge
  - there is added complexity working through the stack



# Building Resilience with AI Chaos

- Creative chaos scenarios for testing resilience
- Step by step instructions to perform the scenarios
- Tailor scenarios for your environment
- Suggested metrics to look at
- Suggested remediation procedures

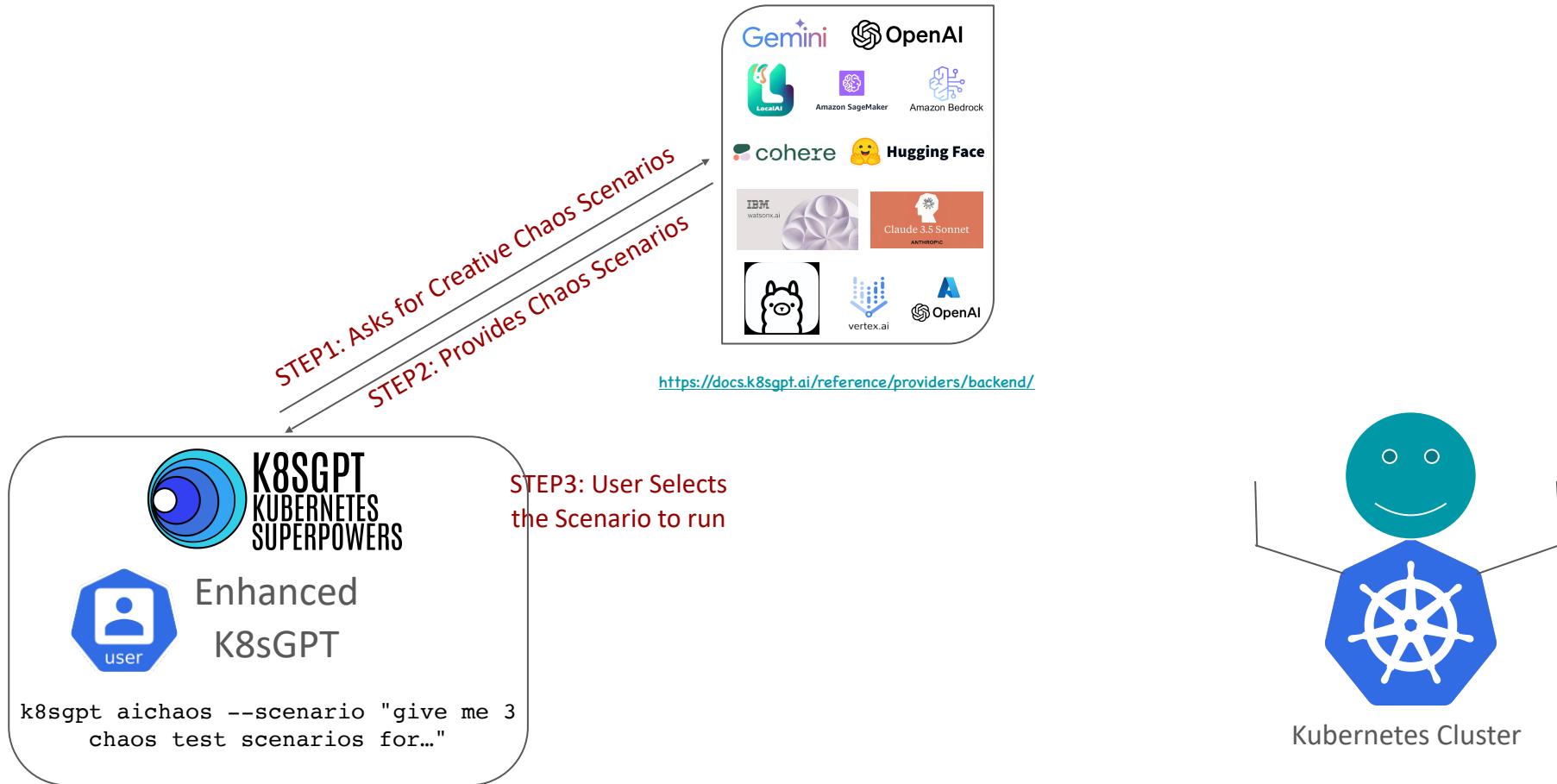
Enhance chaos experiments using AI



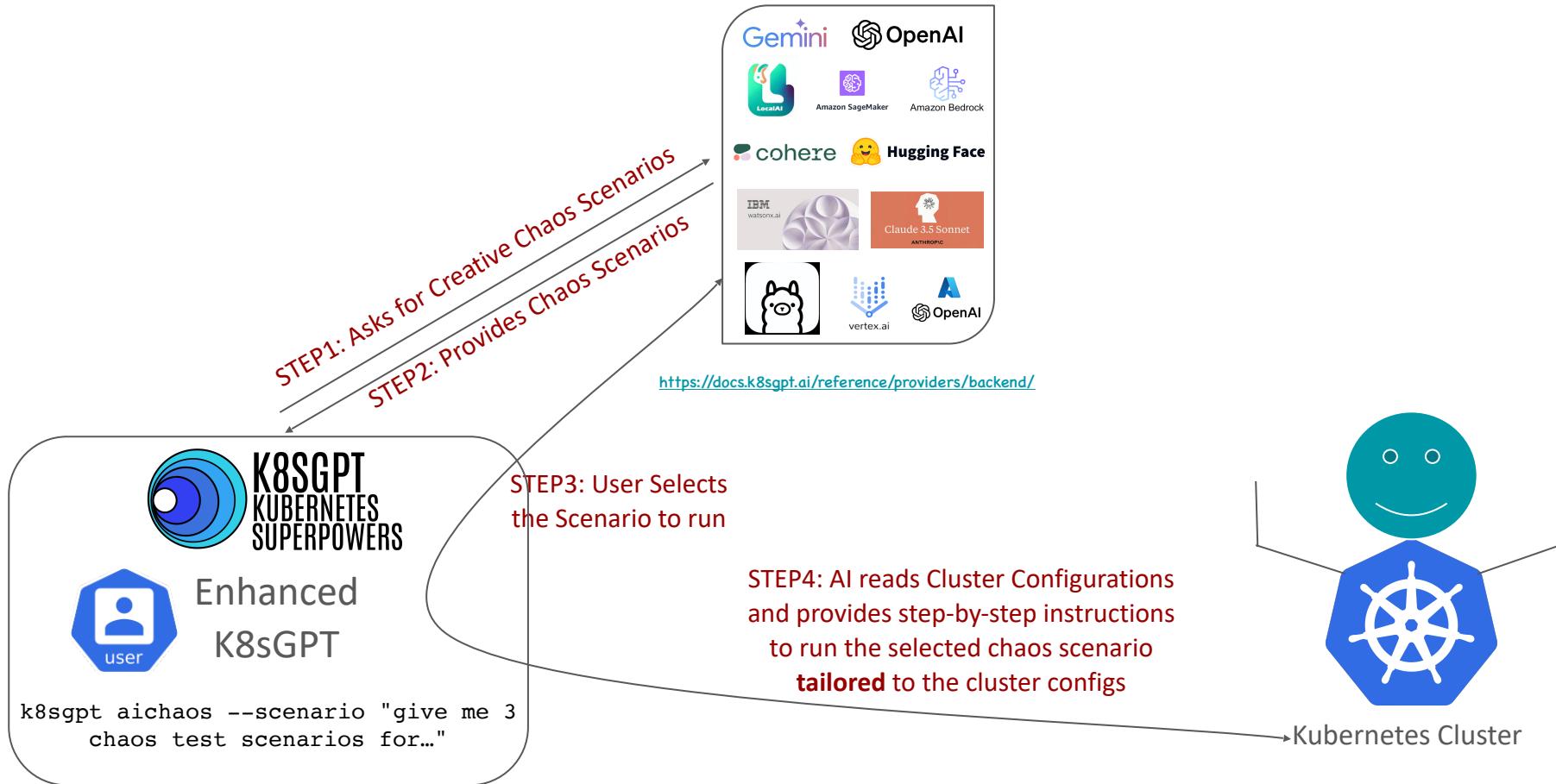
# Building Resilience with AI Chaos



# Building Resilience with AI Chaos



# Building Resilience with AI Chaos



```
51 func (c *ClaudeClient) GetCompletion(ctx context.Context, prompt string) (string, error) {
52
53     req := anthropic.MessagesRequest{
54         Model:      anthropic.ModelClaude3Dot5Sonnet20240620,
55         System:     "You are a Kubernetes networking and Gateway API expert. You are have vase experience with chaos tests and stress tests",
56         Temperature: ptr.To(c.temperature),
57         TopP:       ptr.To(c.topP),
58         TopK:       ptr.To[int](int(c.topK)),
59         MaxTokens:  maxToken,
60     }
}
```

# AI Chaos Testing: Scenario 1

## Route Conflicts



```
k8sgpt aichaos --backend claude --scenario "give me 3 chaos
tests scenarios to test the resiliency of my HTTPRoutes
configs"
```

# Scenario 1 - Routes Chaos

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: istio
  listeners:
  - name: http
    protocol: HTTP
    port: 80

apiVersion: v1
kind: Service
metadata:
  name: infra-backend-v1
spec:
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 3000
  selector:
    app: infra-backend-v1
```

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: my-route
spec:
  parentRefs:
  - name: my-gateway
  hostnames:
  - "lior.example.com"
  rules:
  - matches:
    - path:
      value: /
  backendRefs:
  - name: infra-backend-v1
    port: 8080
  - matches:
    - path:
      value: /shop
  backendRefs:
  - name: infra-backend-v1
    port: 8080
```

llorieberman@lieberman: ~/github/tssurya-k8sgpt

llorieberman@lieberman: ~/github/tssurya-k8sgpt (uplink-keychain)

llorieberman@lieberman: ~/github/tssurya-k8sgpt (uplink-keychain)

llorieberman@lieberman: ~/github/tssurya-k8sgpt\$

llorieberman@lieberman: ~(zsh)

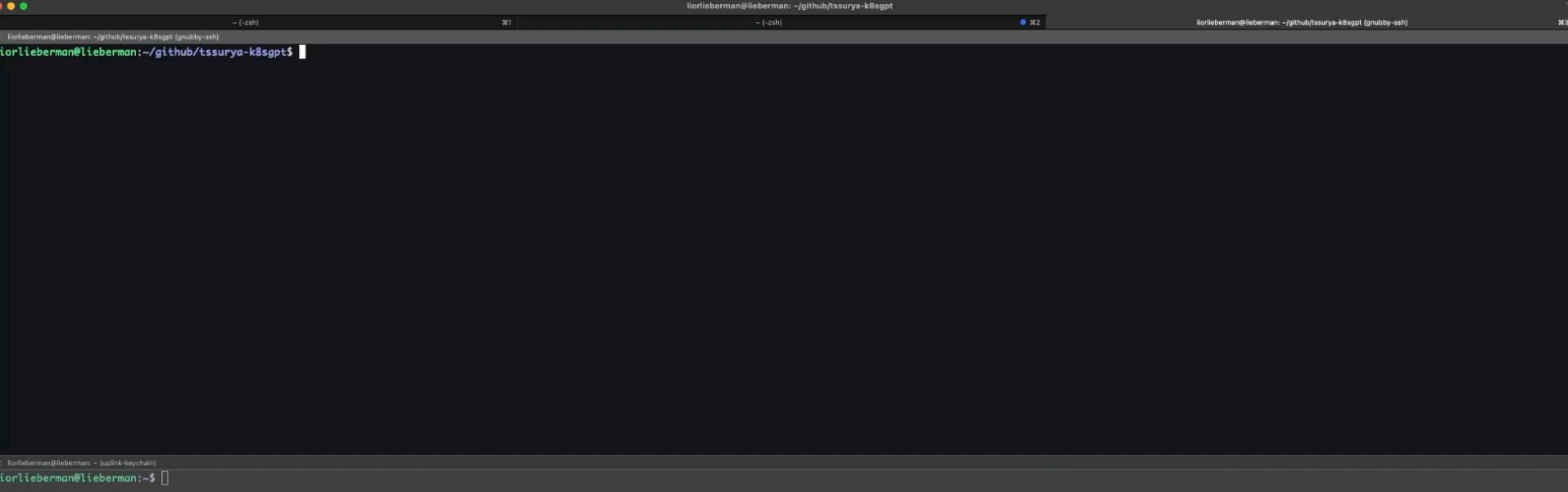
llorieberman@lieberman: ~(zsh)

llorieberman@lieberman: ~(zsh)

llorieberman@lieberman: ~(gnubby-ssh)

llorieberman@lieberman: ~\$

*k8sgpt aichaos --scenario="give me 3 chaos tests scenarios  
to test the resiliency of my HTTPRoutes configs"*



liorieberman@lieberman: ~ (uplink-keychain)  
liorieberman@lieberman:~\$

*“look at my cluster's HTTPRoutes, please give me the corresponding YAMLs and kubectl commands to perform scenario 1. make sure it is relevant to my current applied HTTPRoute and Service resources. All templates and commands should target the route-chaos namespace.”*

```
llorlieberman@lieberman: ~/github/tssurya-k8sgpt
```

- Create a new Deployment, Service, and HTTPRoute (Route B) with identical matching criteria as Route A.  
- Suddenly delete Route A and quickly recreate it.  
- Observe that traffic doesn't restore to the original backend (Route A) as expected, but instead goes to Route B's backend due to the older creation timestamp.

This scenario tests the gateway's behavior during rapid changes and highlights the importance of creation timestamps in routing decisions.

2. Rapid Scale Down and Up of Backend Pods:

- Set up multiple HTTPRoutes pointing to different backend services.
- During peak traffic, rapidly scale down the pods of one backend service to zero.
- Observe how the gateway handles the sudden unavailability of endpoints.
- Quickly scale the pods back up to their original number.
- Monitor how traffic is redistributed and if there's any downtime or errors during the process.

This test evaluates the gateway's ability to handle sudden changes in backend availability and its traffic distribution mechanisms.

3. Network Partition Between Gateway and Backend Services:

- Set up multiple HTTPRoutes with backend services in different network segments.
- Simulate a network partition that isolates the gateway from a subset of backend services.
- Introduce new HTTPRoutes during the partition.
- Restore network connectivity.
- Observe how the gateway recovers, if it correctly routes to all available services, and how it handles the newly introduced routes.

This scenario tests the gateway's resilience to network issues and its ability to recover and incorporate new routing rules after a major disruption.

These chaos tests will help you identify potential weaknesses in your HTTPRoute configurations and gateway implementation, ensuring better resilience in production environments.

```
llorlieberman@lieberman: ~$ 
```

2. Now, let's create a new HTTPRoute for v3:

```
```yaml
# v3-httproute.yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: my-route-v3
  namespace: route-chaos
spec:
  hostnames:
    - lior.example.com
  parentRefs:
    - name: my-gateway
  rules:
    - backendRefs:
        - name: infra-backend-v3
          port: 8080
      matches:
        - path:
            type: PathPrefix
            value: /
    - backendRefs:
        - name: infra-backend-v3
          port: 8080
      matches:
        - path:
            type: PathPrefix
```

liorlieberman@lieberman:~ (uplink-keychain)

```
liorlieberman@lieberman:~$ vim v3-deployment-service.yaml
liorlieberman@lieberman:~$ kubectl apply -f v3-deployment-service.yaml
deployment.apps/infra-backend-v3 created
service/infra-backend-v3 created
liorlieberman@lieberman:~$ vim v3-httproute.yaml
liorlieberman@lieberman:~$
```

2. Now, let's create a new HTTPRoute for v3:

```
```yaml
# v3-httproute.yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: my-route-v3
  namespace: route-chaos
spec:
  hostnames:
  - lior.example.com
  parentRefs:
  - name: my-gateway
  rules:
  - backendRefs:
    - name: infra-backend-v3
    port: 8080
    matches:
    - path:
        type: PathPrefix
        value: /
  - backendRefs:
    - name: infra-backend-v3
    port: 8080
    matches:
    - path:

```

liorieberman@lieberman:~(uplink-keychain)

```
liorieberman@lieberman:~$ kubectl exec -it curl -- curl -v --header "Host: lior.example.com" http://my-gateway-istio/shop | grep infra
"infra-backend-v1.route-chaos.svc.cluster.local:8080/*"
"pod": "infra-backend-v1-59bd6c847f-rpbzl"
liorieberman@lieberman:~$ kubectl exec -it curl -- curl -v --header "Host: lior.example.com" http://my-gateway-istio/shop | grep infra
"infra-backend-v1.route-chaos.svc.cluster.local:8080/*"
"pod": "infra-backend-v1-59bd6c847f-rpbzl"
^[[Aliorieberman@lieberman:~$ kubectl exec -it curl -- curl -v --header "Host: lior.example.com" http://my-gateway-istio/shop | grep infra
"infra-backend-v1.route-chaos.svc.cluster.local:8080/*"
"pod": "infra-backend-v1-59bd6c847f-rpbzl"
liorieberman@lieberman:~$
```

```
llorieberman@lieberman:~(zsh) ~ (-zsh) llorieberman@lieberman:~(-zsh) llorieberman@lieberman:~(uplink-keychain) ~ (-zsh) llorieberman@lieberman:~/github/tssurya-k8sgpt (grubby-ssh)
```

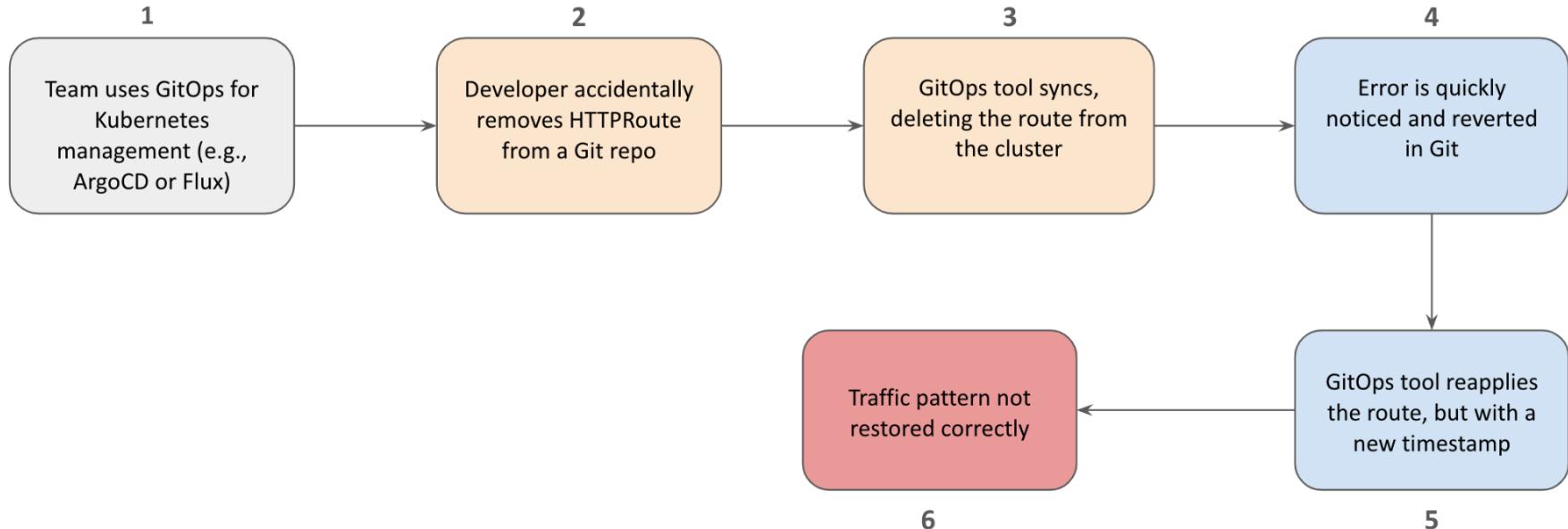
matches:  
- path:  
 type: PathPrefix  
 value: /shop  
...  
  
Apply this with:  
```  
kubectl apply -f v3-httproute.yaml  
```  
  
3. Now, let's perform the chaos test:  
```bash  
# Delete the original HTTPRoute  
kubectl delete httproute my-route -n route-chaos  
  
# Quickly recreate the original HTTPRoute  
cat <<EOF | kubectl apply -f -  
apiVersion: gateway.networking.k8s.io/v1  
kind: HTTPRoute  
metadata:  
 name: my-route  
 namespace: route-chaos  
spec:  
 hostnames:  
 - lior.example.com  
 parentRefs:  
 - name: my-gateway  
  
llorieberman@lieberman:~(uplink-keychain)  
llorieberman@lieberman:~\$ kubectl apply -f v3-httproute.yaml  
httproute.gateway.networking.k8s.io/my-route-v3 created  
llorieberman@lieberman:~\$ kubectl exec -it curl -- curl -v --header "Host: lior.example.com" http://my-gateway-istio/shop | grep infra  
 "infra-backend-v1.route-chaos.svc.cluster.local:8080/\*"  
 "pod": "infra-backend-v1-59bd6c847f-rpbzl"  
llorieberman@lieberman:~\$ kubectl exec -it curl -- curl -v --header "Host: lior.example.com" http://my-gateway-istio/shop | grep infra  
 "infra-backend-v1.route-chaos.svc.cluster.local:8080/\*"  
 "pod": "infra-backend-v1-59bd6c847f-rpbzl"  
llorieberman@lieberman:~\$



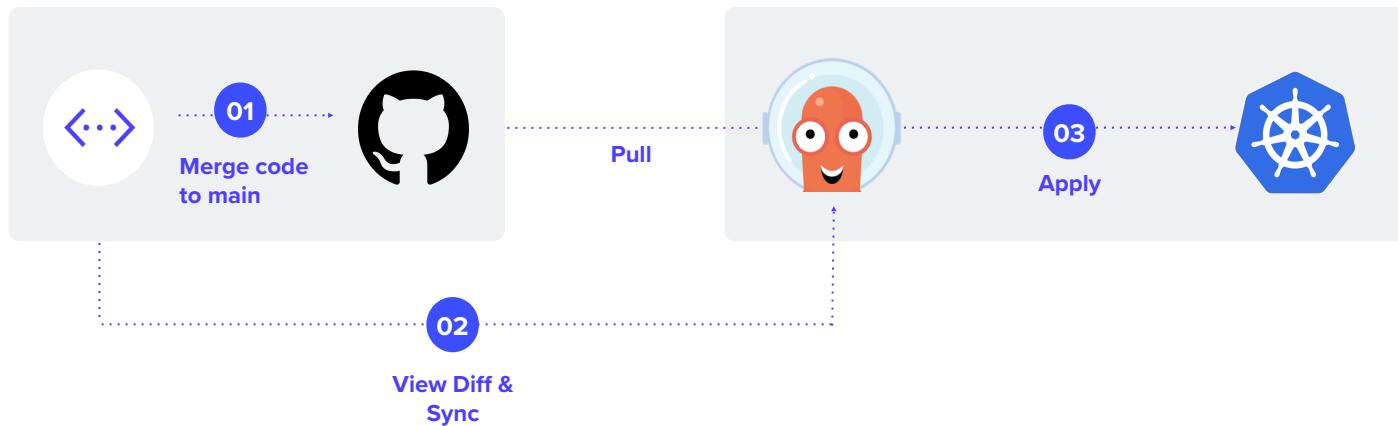
```
litorieberman@lieberman: ~ (zsh) 11 - (-zsh) 22 litorieberman@lieberman: ~ (uplink-keychain) 23
x litorieberman@lieberman: ~/github/tssurya-k8sgpt (grubby-ssh)
- name: infra-backend-v1
  port: 8080
  matches:
  - path:
    type: PathPrefix
    value: /
- backendRefs:
- name: infra-backend-v1
  port: 8080
  matches:
  - path:
    type: PathPrefix
    value: /shop
EOF
```
4. Observe the routing behavior:
```bash
# Check the creation timestamps of the HTTPRoutes
kubectl get httproutes -n route-chaos --sort-by=.metadata.creationTimestamp
# Test the routing (replace <GATEWAY_IP> with your gateway's external IP)
curl -H "Host: lior.example.com" http://<GATEWAY_IP>/
curl -H "Host: lior.example.com" http://<GATEWAY_IP>/shop
```
You should observe that the traffic is not restored to the original backend (infra-backend-v1) as expected, but instead goes to infra-backend-v3's backend due to the older creation timestamp of my-route-v3.

litorieberman@lieberman: ~ (uplink-keychain)
litorieberman@lieberman: ~ $ kubectl get httproutes -n route-chaos --sort-by=.metadata.creationTimestamp
NAME          HOSTNAMES      AGE
my-route-v3   ["lior.example.com"]  49s
my-route      ["lior.example.com"]  17s
litorieberman@lieberman: ~ $
```

# This Happens in Real World Setups



# This Happens in Real World Setups



```
func (c *ClaudeClient) GetCompletion(ctx context.Context, prompt string) (string, error) {
51
52    req := anthropic.MessagesRequest{
53        Model:      anthropic.ModelClaude3Dot5Sonnet20240620,
54        System:     "You are a Kubernetes Networking expert. You also have vast experience with Litmus Networking chaos tests and stress tests.",
55        Temperature: ptr.To(c.temperature),
56        TopP:       ptr.To(c.topP),
57        TopK:       ptr.To[int](int(c.topK)),
58        MaxTokens:   maxToken,
```

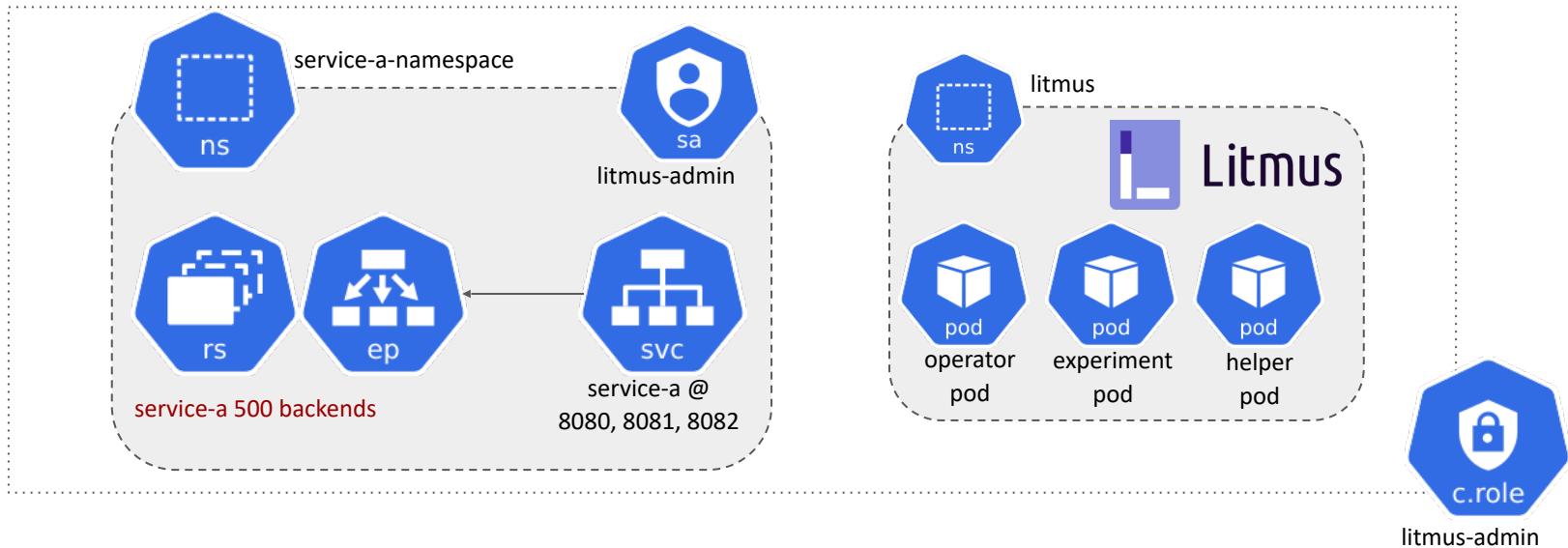
# AI Chaos Testing: Scenario 4

## Service Latency at Scale



```
k8sgpt aichaos --backend claude --scenario "give me 3
chaos test scenarios to test the resiliency of my
Network"
```

# Service Latency at Scale: DEMO: Setup



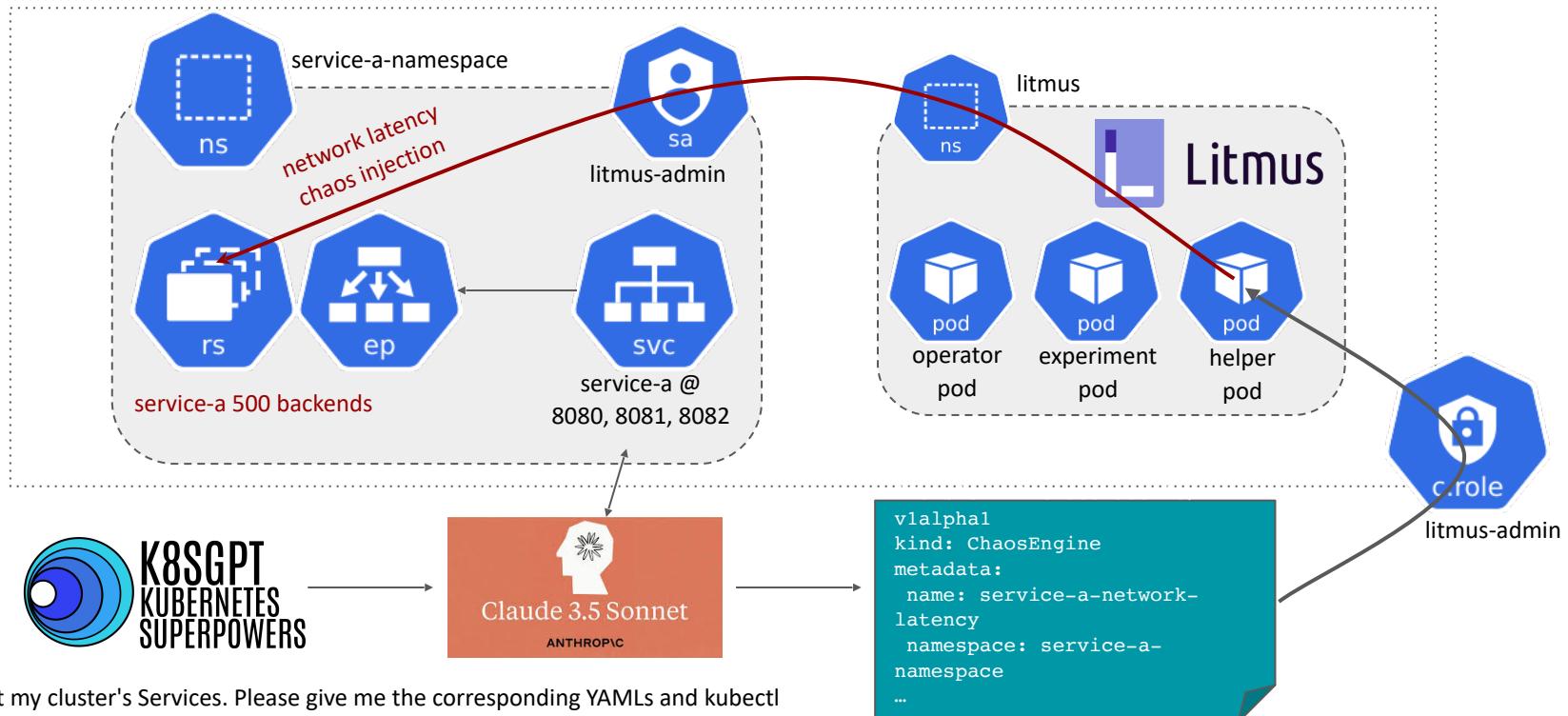


surya@fedora:~/kubecon-na-demo-scenario-4\$

surya@fedora:~/kubecon-na-demo-scenario-4



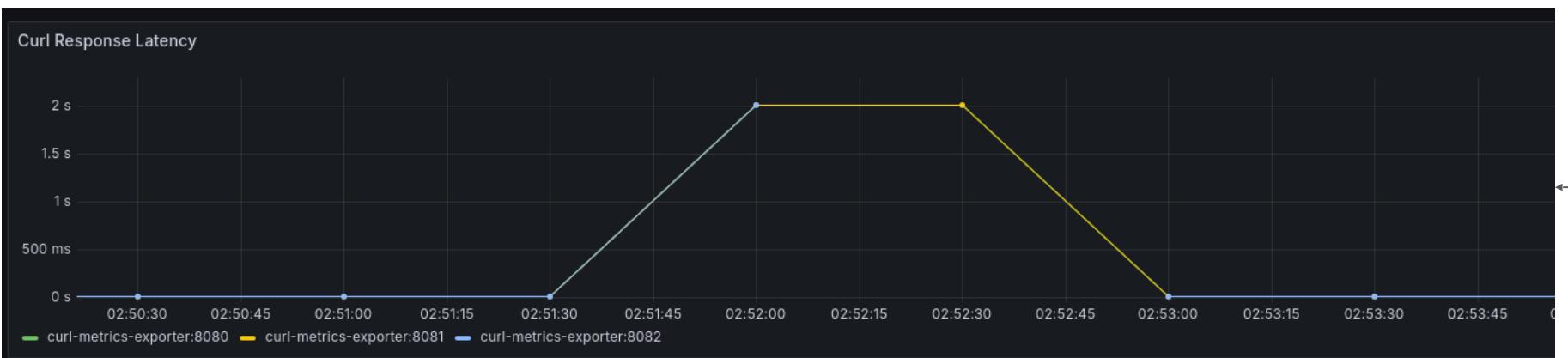
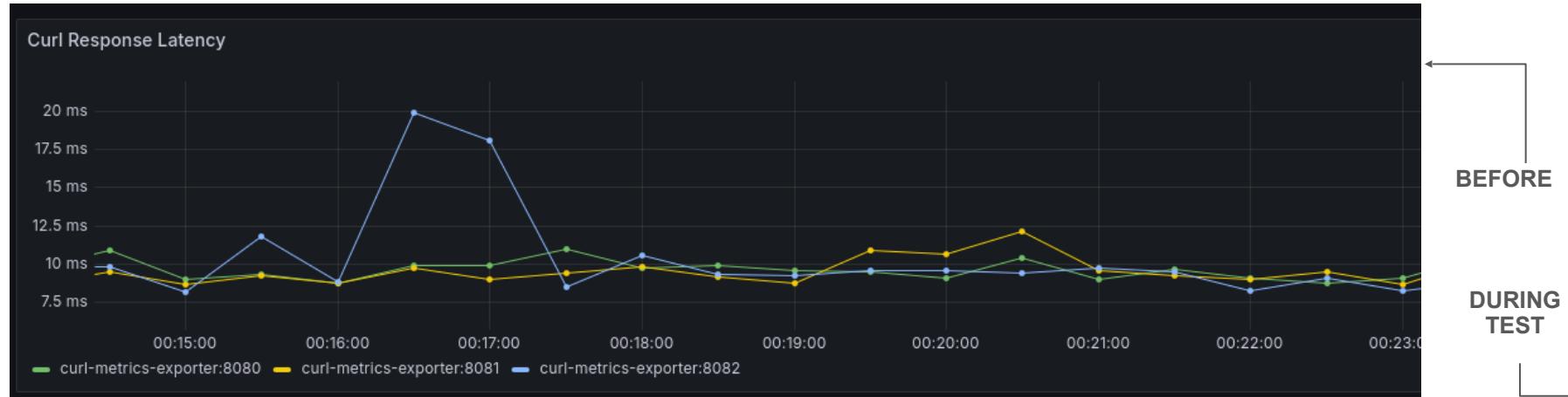
# Service Latency at Scale: DEMO: AI Chaos Injection



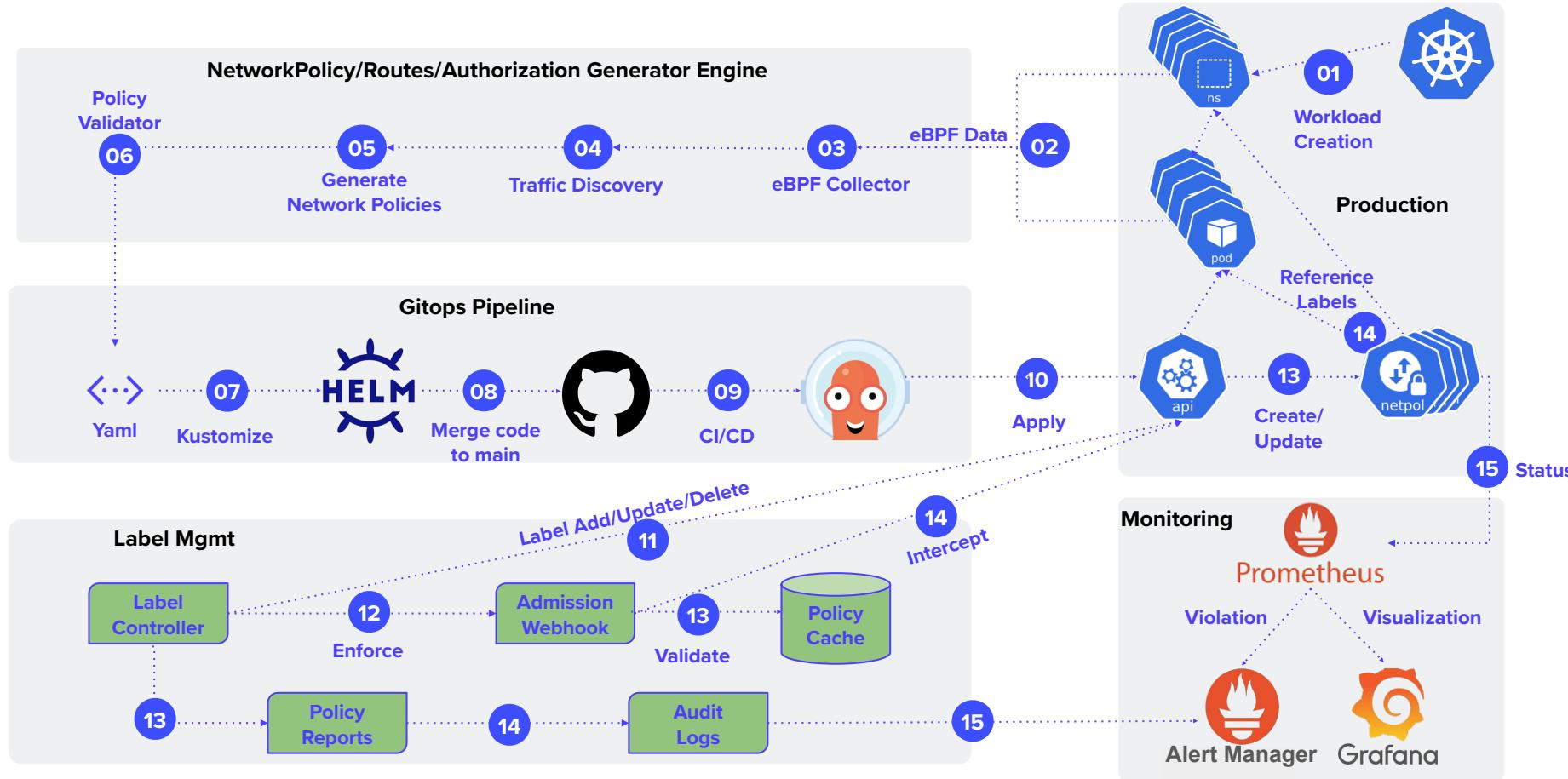
look at my cluster's Services. Please give me the corresponding YAMLs and kubectl commands to perform network latency in scenario 2. Let us inject a latency of 1000ms for 90 seconds targeting 50% of pods. Name of my service account is "litmus-admin". Make sure it is relevant to my applied Service resources. All templates and commands should target the service-a-namespace.



# Scenario 4: Service Latency at Scale: DEMO



# Production Environments are More Complex



# Lessons Learned

# Lessons Learned: Precautions when using AI

## Things to keep in mind

- Accuracy needs improvement - Don't trust it blindly
- Context is required - enough context is key for optimal results
- The models are not deterministic
- Protect your data - make sure you understand implications of using a free subscription
- Have a well-defined emergency stop procedure - AI is no exception

# Lessons Learned: Why use AI for these scenarios

## Advantages

- Can process large context and tailor scenarios accordingly
- Enhances test diversity and creativity
- Handles complex abstractions for users
- Has richer knowledge base than a human has
  - Example uses chaos mesh and litmus to simulate chaos
- Saves plenty of time!

# Wrapping up

- Ecosystem is rapidly changing
- Use AI tools, provide feedback to make them better
- Check out our [POC](#)





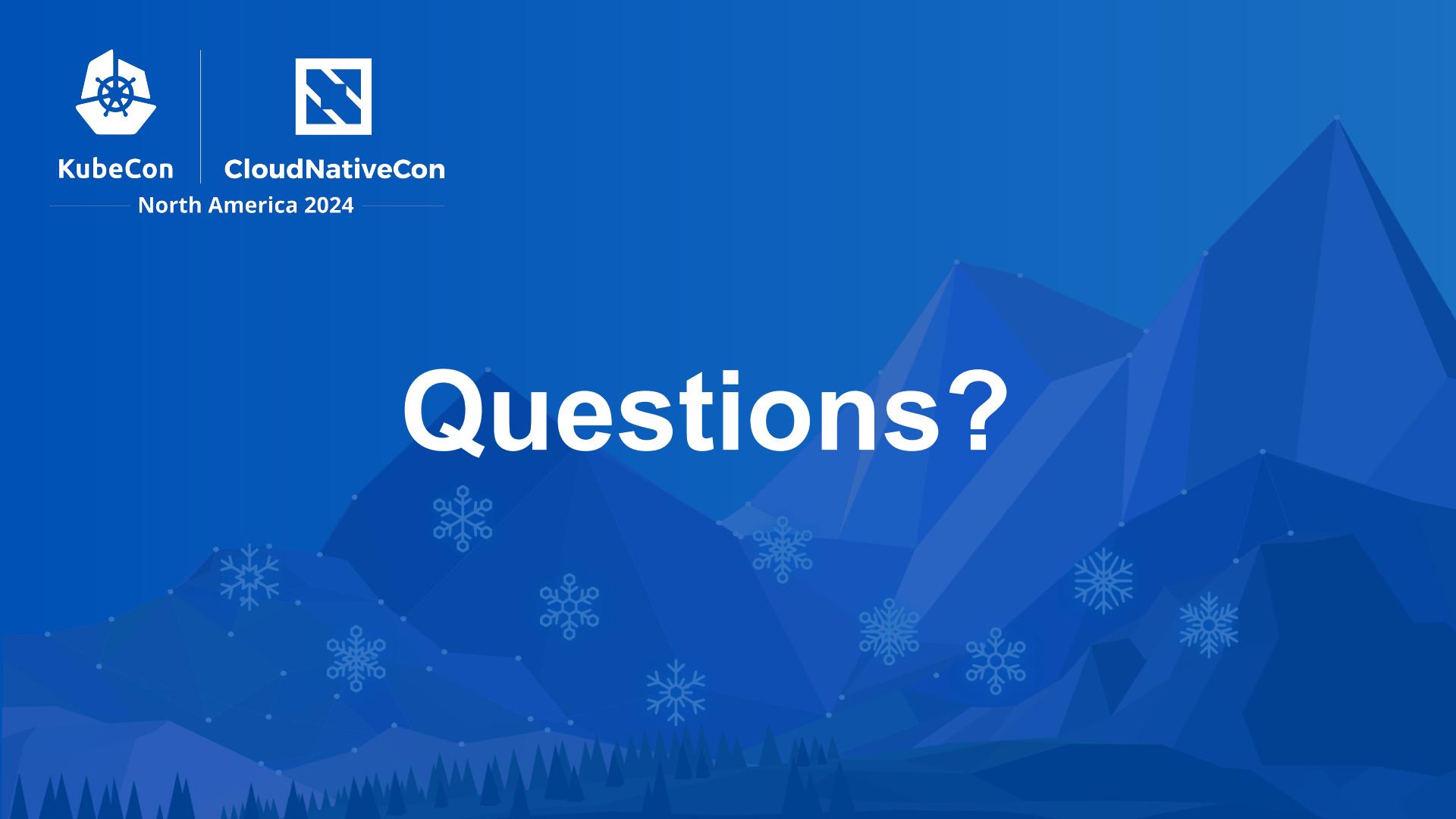
KubeCon



CloudNativeCon

North America 2024

# Questions?





KubeCon



CloudNativeCon

North America 2024